TEKNILLINEN KORKEAKOULU
Elektroniikan, tietoliikenteen ja automaation tiedekunta
Sähkötekniikan laitos

Jussi Rintamäki

Koneenohjausyksikön laiteohjelmiston automaattinen testaus

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi
diplomi-insinöörin tutkintoa varten Espoossa 2.12.2008.

Työn valvoja professori Seppo Ovaska

Työn ohjaaja diplomi-insinööri Mika Lammila

Teknillinen korkeakoulu
Diplomityön tiivistelmä

Tekijä: Jussi Rintamäki

Työn nimi: Koneenohjausyksikön laiteohjelmiston automaattinen testaus

Päivämäärä: 2.12.2008 Sivumäärä: 69

Tiedekunta: Elektroniikan, tietoliikenteen ja automaation tiedekunta

Professuuri: S-81 Teollisuuselektroniikka

Työn valvoja: Professori Seppo Ovaska

Työn ohjaaja: Diplomi-insinööri Mika Lammila

Älykkäiden koneenohjausyksiköiden ja niistä koostuvien ohjausjärjestelmien monimutkaistuessa myös niiden testauksesta on tullut varsin vaativa tehtävä. Ohjausyksiköiden laiteohjelmiston kattava testaus on muodostunut aikaa vieväksi ja haastavaksi tehtäväksi, jota on jo lähes mahdotonta suorittaa manuaalisesti yksittäisiä testejä suorittamalla. Tässä työssä on tarkoitus kehittää koneenohjausyksiköiden laiteohjelmiston testaukseen automaattinen testausjärjestelmä, jonka avulla manuaalista testausta voitaisiin huomattavasti vähentää.

Työ koostuu viidestä pääosasta. Ensimmäiseksi esitellään ongelma ja lähtökohdat, joiden pohjalta työtä on lähdetty toteuttamaan. Toisessa osassa esitellään kehitettävän testausjärjestelmän testausobjekti eli ohjausyksikkö. Tämän jälkeen perehdytään työn kannalta olennaiseen testauksen teoriaan ja testauksen automatisointiin sekä itse testausjärjestelmään. Työn neljännessä pääosassa esitellään koko työn eteneminen suunnittelusta testausjärjestelmän toteutukseen ja testaukseen. Lisäksi neljännessä osassa perehdytään järjestelmän ylläpitoon ja elinkaareen. Viidennessä eli viimeisessä osassa tarkastellaan työn lopputulosta eli testausjärjestelmää alkuperäisten vaatimusten ja lähtökohtien pohjalta.

Työn lopputuloksena syntyi automaattinen ja modulaarinen testausjärjestelmä, jolla voidaan testata kaikkia koneenohjausyksikön laiteohjelmiston olennaisia perustoimintoja nopeasti ja vaivattomasti alkuperäisten tavoitteiden mukaisesti. Järjestelmä ratkaisee suurimmaksi osaksi manuaalisen testauksen tuottamat ongelmat, joita ovat muun muassa testien hitaus ja huono toistettavuus.

Avainsanat: Koneenohjausjärjestelmä, ohjausyksikkö, testaus, automatisointi, laiteohjelmisto, PLC, PXI, IEC 61131, LabVIEW, CAN, CANopen

Helsinki University of Technology
Abstract of the Master's Thesis

Author: Jussi Rintamäki

Title of the Thesis: Automatic testing of machine control unit firmware

Date: 2.12.2008 Number of pages: 69

Faculty: Faculty of Electronics, Communications and Automation

Professorship: S-81 Industrial Electronics

Supervisor: Professor Seppo Ovaska

Instructor: Master of Science Mika Lammila

Intelligent machine control units and systems based on these units are getting more complicated day by day. Testing of these units and systems has become very demanding because of the level of their complexity. Comprehensive testing of control units' firmware has become a time consuming and demanding task that is almost impossible to do with manual testing practices. The aim of this assignment is to develop an automatic testing system for control units' firmware testing that would reduce the need for manual testing.

There are five main parts in this assignment. The problem and the base of this assignment are introduced in the first part. In the second part there is an introduction of the test object of this work, which is the intelligent control unit and its firmware. The third part introduces the main aspects of testing and automation according to this assignment. The third part also contains a review of the hardware of the testing system. The fourth part demonstrates the entire execution of the testing system. The end of the fourth part concentrates on the testing system's lifecycle and maintenance. In the fifth and final part of the assignment is the conclusion with consideration towards the original demands and the core of the assignment.

As a result, an automatic testing system was developed in this assignment. The testing system can be used for fast and reliable testing of all the main functionalities of the control units' firmware according to what was defined at the beginning of this assignment. The testing system resolves all the main problems of manual testing, which are for example, slow execution and poor repetition of the tests.

Keywords: Machine control system, control unit, testing, automatization, firmware, PLC, PXI, IEC 61131, LabVIEW, CAN, CANopen

Alkusanat

Tämä diplomityö on tehty Seinäjoella Epec Oy:n tuotekehitysosastolla. Työn ohjaajana on toiminut diplomi-insinööri Mika Lammila, jota haluan kiittää kaikesta saamastani avusta. Erityisesti haluan kiittää työn valvojaa professori Seppo Ovaskaa hänen osoittamastaan mielenkiinnosta työtäni kohtaan sekä korvaamattoman arvokkaasta kannustuksesta hetkinä, joina minusta on tuntunut, että työ ei tule koskaan valmistumaan.

Suuret kiitokset ansaitsevat myös perheeni ja ystäväni antamastaan tuesta ja kannustuksesta diplomityöni ja opintojeni aikana.

Seinäjoella 2.12.2008

Jussi Rintamäki

Sisällysluettelo

Termi-, lyhenne- ja symboliluettelo

Termit ja lyhenteet

AMPSEAL8	8-pinninen liitin (uros-naaras)
AMPSEAL23	23-pinninen liitin (uros-naaras)
CAN	Sarjamuotoinen tietoliikenneväylä
CANopen	Fyysisen CAN-väylän päälle rakennettu korkeamman tason kommunikaatioprotokolla
COB-ID	Communication Object Identifier; CANopenissa CAN-kehyksen 11-bittinen tunnistekenttä
CoDeSys	IEC 61131-3 standardin mukaisten sovellusten kehitykseen tarkoitettu sovelluskehitysympäristö
C-kieli	Imperatiivinen ohjelmointikieli
Darlington Driver	Darlington kytkentä
EMCY	CANopenin protokolla, joka ilmaisee laitteen sisäisiä vikoja
Epec 2024	Epec Oy:n kehittämä koneenohjausyksikkö
Error Control	CANopenin protokolla, jota käytetään virhetilanteiden havainnointiin ja korjaamiseen
Ethernet	Pakettipohjainen lähiverkkoratkaisu
Firmware	Sama kuin laiteohjelmisto
Flash	Haihtumaton uudelleenohjelmoitava muisti
GPIB	General purpose interface bus; erityisesti tietokoneiden, lisälaitteiden ja laboratorioinstrumenttien väliseen kommunikaatioon suunniteltu rinnakkaismuotoinen dataväylä
G-kieli	LabVIEW-ohjelmointiympäristön ohjelmointikieli
IEC 61131	Avoin standardi ohjelmoitavien logiikoiden arkkitehtuurille
IEC 61158	Kenttäväylä standardi
I/O	Input/Output; Tulo/Lähtö
LabVIEW	Ohjelmointiympäristö, joka perustuu graafiseen G-kieleen
LabVIEW Real-Time	Reaaliaikakäyttöjärjestelmä
Laiteohjelmisto	Laitteen haihtumattomaan muistiin tallennettu ohjelmisto, joka toimii rajapintana laitteiston ja sovellusohjelmiston välillä
Multiplekseri	Laite, jolla voidaan ohjata yksi useasta sisään tulevasta signaalista laitteen ulostuloon
NMT	CANopenin protokolla, jolla hallitaan CANopen-laitteiden toimintaa
PC	Personal Computer; tietokone

PCI	Peripheral Component Interconnect; alun perin Intelin kehittämä, myöhemmin PCI-standardin mukainen rinnakkaismuotoinen tietokoneväylä oheislaitteiden liittämiseksi tietokoneisiin
PCI Express	Sarjamuotoinen tiedonsiirtoväylä oheislaitteiden liitämiseksi tietokoneisiin
PDO	CANopenin protokolla, jonka avulla voidaan siirtää sovellustietoja verkon laitteiden välillä
PLC	Programmable Logic Controller; ohjelmoitava logiikka
Pulse Out	Pulssilähtö; pulssimuotoista signaalia tuottava lähtö
PWM	Pulse-Width Modulation; pulssinleveysmodulointi
PXI	CompactPCI:in eli teollisuus ja tutkimuslaitoskäyttöön koteloidun PC-tekniikan versio. Väylä tekniikkana PCI
PXIe	Sama kuin PXI, mutta PCI-väylä on korvattu PCI Express-väylällä
RAM	Random Access Memory; keskusmuisti (työmuisti)
RS-232	Standardi, joka määrittelee kahden tietokonelaitteen välisen sarjamuotoisen datasignaalikommunikaation
RS-485	Standardi, joka määrittelee differentiaalisen sarjaväylän sähköiset ominaisuudet
SDO	CANopenin protokolla, jota käytetään verkossa olevien CANopen-laitteiden objektikirjastojen arvojen lukemiseen ja kirjoittamiseen
SYNC	CANopenin protokolla, joka synkronoi CAN-verkon toimintaa
Testausobjekti	Testauksen kohde. Tässä työssä ohjausyksikkö/ohjausyksikön laiteohjelmisto
Testisekvenssi	Määritelty järjestys ajaa testiohjelmat
TestStand	Testien hallintaan tarkoitettu ohjelmisto
TIA-232-F	Sama kuin RS-232
TIA-485-A	Sama kuin RS-485
TimeStamp	CANopenin protokolla, joka asettaa CAN-verkkoon yhtenäisen ajan
USB	Universal Serial Bus; sarjaväyläarkkitehtuuri oheislaitteiden liittämiseksi tietokoneisiin
Virtuaali-instrumentti	LabVIEW-ohjelma
Voltage Regulator	Jänniteregulaattori, jännitteentasaaja

Symbolit

A	Pulssinleveysmoduloidun signaalin maksimiarvo
A', A'', A'/A''	RS-485-väylän ei-invertoivat pinnit
AO/I	Analoginen virtalähtö
AO/V	Analoginen jännitelähtö
B', B'', B'/B''	RS-485-väylän invertoivat pinnit
CAN_H	CAN_High; CAN-väylän korkeampi potentiaali
CAN_L	CAN_Low; CAN-väylän matalampi potentiaali
COM	Common; maapotentiaali
DC	Direct Current; tasavirta
DI	Digital Input; digitaalinen tulo
DO	Digital Output; digitaalinen lähtö
DO/Sink	Maapotentiaaliin kytkevä digitaalinen lähtö
DO/Source	Jännitteeseen kytkevä digitaalinen lähtö
GND	Ground; maapotentiaali
LOAD	Kuorma
P	I/O-kortin kytkentäpiste
Rxd	Received Data; signaali RS-232-väylässä
Txd	Transmitted Data; signaali RS-232-väylässä
V_{eff}	Pulssinleveysmoduloidun signaalin keskiarvo
VCC	Positiivinen käyttöjännite
X	Digitaalisen tulon prosessorin puoleinen kytkentäpiste

1. Johdanto

Nykyaikaisissa työkoneissa, kuten esimerkiksi metsä- ja kaivostyökoneissa, on valtavasti erilaisia älykästä logiikkaa vaativia toimintoja. Älykkyys koneiden yhteydessä tarkoittaa koneen kykyä tuottaa jokin haluttu looginen vaste tietyn herätteen sattuessa. Tällainen älykkyys toteutetaan nykyään helposti prosessoreiden ja niiden ohjelmoinnin avulla. Prosessoritekniikan kehitys ja tarve tehostaa työkoneiden toimintaa, automatisoimalla erilaisia tehtäviä, ovat aiheuttaneet sen, että nykyään jokaisesta vähänkin suuremmasta työkoneesta löytyy yksi tai useampi sulautettu ohjain, ohjausyksikkö, joka ohjaa koneen toimintoja. Sulautetut järjestelmät, jotka koostuvat useista ohjausyksiköistä, ovat arkipäivää nykypäivän työkoneissa. Ne suorittavat laskentatehoa ja erilaisia tarkkuutta vaativia tehtäviä, kuten moottorin ja venttiilien ohjausta. Sulautettu ohjain suorittaa tehtävät riittävän nopeasti ja tarkasti tarjoamalla työkoneelle ja sen kuljettajalle reaaliaikaisen vasteen, mikä on ominaista sulautetuille järjestelmille.

Työkoneiden erilaisuudesta ja vaihtelevista toimintaympäristöistä johtuen on ollut järkevää kehittää modulaarinen älykäs ohjausjärjestelmä, joka koostuu toisiinsa kytketyistä ohjausyksiköistä. Järjestelmä tarjoaa mahdollisuuden viedä ohjausyksiköt lähelle paikkaa, jossa niitä tarvitaan. Tällaisessa järjestelmässä jokainen yksikkö suorittaa yleensä yhtä tai muutamaa tiettyä tehtävää ja toimii yhdessä muiden järjestelmän osien kanssa jonkin standardin kommunikointirajapinnan kautta. Modulaarinen järjestelmäarkkitehtuuri vähentää suunnitteluvaiheen kompleksisuutta, helpottaa ja parantaa järjestelmän ylläpidettävyyttä sekä olennaisesti pienentää järjestelmäkokonaisuuden haavoittuvuutta jakamalla laajan järjestelmän pienempiin osakokonaisuuksiin, mikä on tärkeä vaatimus työkoneiden turvalliselle ja oikealle toiminnalle. Modulaarisuus tarjoaa myös, yhdessä standardin kommunikointirajapinnan kanssa, mahdollisuuden yhdistää eri valmistajien laitteita yhdeksi kokonaisjärjestelmäksi, koska ohjausyksiköt voivat toimia näennäisesti toisistaan riippumatta.

Ohjausyksiköiden suorittamat toiminnot monimutkaistuvat nopeasti, kun suunnitellaan uusia järjestelmiä, joissa ohjattavien toimintojen määrä on suurempi ja automaation aste korkeampi. Ohjausyksiköt pystyvätkin suorittamaan jo monimutkaisia tehtäviä, kuten esimerkiksi moottorin- tai vaihteiston ohjauksen. Ohjausyksikön monimutkaisuudesta johtuen sen suunnittelu ja ohjelmointi ovat muodostuneet todella vaativiksi tehtäviksi. Tämä aiheuttaa myös yksikön toiminnan testaamiselle uusia haasteita. Nykyään testaus onkin osoittautunut niin vaativaksi ja aikaa vieväksi tehtäväksi, että sitä ei kannata enää järkevästi suorittaa manuaalisesti testitapauksia toistamalla, kuten ennen on voitu vielä jossain määrin järkevästi tehdä.

Tämän työn tarkoituksena on rakentaa älykkään koneenohjausjärjestelmän ohjausyksiköille automaattinen testausjärjestelmä, joka hoitaa automaattisesti ohjausyksiköiden laiteohjelmiston eli firmwaren testauksen. Työn tarkoitus on ensisijaisesti toteuttaa testausjärjes-

telmä, joka nopeuttaa ohjausyksiköiden laiteohjelmiston testaamista ja parantaa testauksen luotettavuutta sekä jäljitettävyyttä.

Työn tavoitteena on, rakentamalla automaattinen testausjärjestelmä, päästä eroon ohjausyksiköiden toistuvasta manuaalisesta testauksesta. Manuaalinen testaus vie paljon aikaa ja yleensä testitapaukset vaihtelevat kerrasta toiseen, johtuen testien suorittajan inhimillisestä vaikutuksesta testaussekvensseihin, jolloin ohjausyksiköiden laiteohjelmiston oikean toiminnan varmistaminen ei saavuta riittävän luotettavaa ja laadukasta tasoa. Manuaalisessa testauksessa tullaan, ohjausyksiköiden toimintojen lisääntyessä ja monimutkaistuessa, väkisin lopulta pisteeseen, jolloin luotettavuuden ja laadun varmistamiseksi vaadittavien testauskombinaatioiden ja erilaisten sekvenssien määrä on niin suuri, että manuaalisten testisekvenssien automatisointi on ainoa vaihtoehto.

Työssä määritellään myös joustavat ja järkevät testitapaukset ohjausyksiköiden toimintojen testaamiseen niin, että laiteohjelmiston eri ominaisuudet tulevat testatuksi kattavasti käytännön toimintaa silmälläpitäen. Testauslaitteisto toteutetaan modulaarisen PXI-arkkitehtuurin pohjalta kytkemällä tarvittavat ohjauskortit PXI-kehikkoon. PXI-arkkitehtuurin käyttö mahdollistaa laitteiston helpon ja joustavan päivityksen tulevaisuudessa, mikä on myös keskeinen tavoite työlle. Testauslaitteiston ohjelmointi tehdään LabVIEW-ohjelmointiympäristön G-kielellä. Testausvaiheiden automatisointiin sekä automaattiseen raportointiin käytetään TestStand-ohjelmistoa.

Tässä työssä esitetään ensin ongelma ja lähtökohdat, joiden ympärille itse työ ja tutkimus rakentuvat. Tämän jälkeen käydään läpi nykyaikaisen älykkään koneenohjausjärjestelmän perusteet ja rakenne sekä perehdytään järjestelmän ohjausyksiköiden rajapintoihin ulkomaailman kanssa. Ohjausyksiköiden rajapinnoista siirrytään tarkastelemaan tarkemmin työn toista ääripäätä eli testausjärjestelmää ja automaattista testausta yleisesti. Lopuksi esitetään työn toteutus ja tulokset sekä tarkastellaan alkuperäisiä tavoitteita verraten niitä lopulliseen automaattiseen testausjärjestelmään.

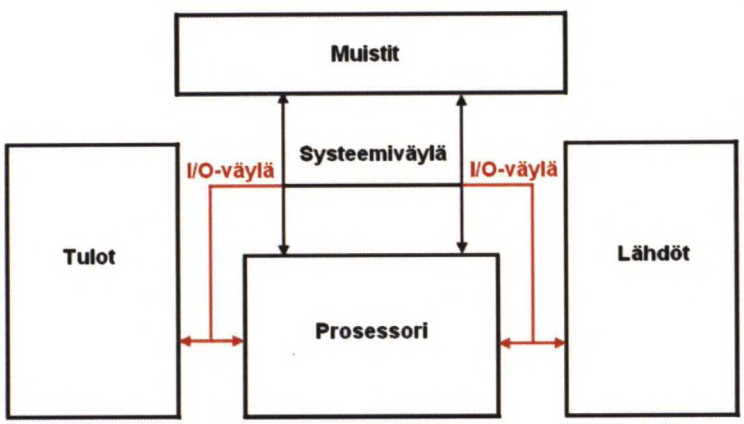
2. Ongelma ja lähtökohdat

Tässä luvussa määritellään tekijät, joiden pohjalta työtä on lähdetty viemään eteenpäin. Ensin määritellään ongelma, johon työllä halutaan tuottaa mahdollisimman mielekäs ratkaisu eli automaattinen laiteohjelmiston testausjärjestelmä, joka suorittaa kaikki määritellyt testit automaattisesti. Seuraavaksi esitellään työn lähtökohdat ja määritellään työn tavoitteet ongelmaa silmälläpitäen. Lopuksi käydään läpi työlle ja sen lopputulokselle asetetut vaatimukset sekä alun perin diplomityön kannalta tarpeellisiksi katsotut rajaukset.

2.1. Ongelma

Älykkäiden koneenohjausyksiköiden ja niistä koostuvien järjestelmien monimutkaistuessa myös niiden testauksesta on tullut varsin vaativa tehtävä. Ohjausyksiköiden laiteohjelmiston kattava testaus on muodostunut aikaa vieväksi ja haastavaksi tehtäväksi, jota on jo lähes mahdotonta suorittaa manuaalisesti yksittäisiä testejä suorittamalla. Ongelma, jonka pohjalle tämä työ rakentuu, on ohjausyksiköiden laiteohjelmiston testaukselle asetettavat laatu ja toistettavuusvaatimukset, joita ei saavuteta enää nykyisellä manuaalisella testauksella.

Kuvassa 1 on esitetty älykkään ohjausyksikön perusarkkitehtuuri. Älykäs ohjausyksikkö rakentuu prosessorin ympärille, mikä on koko laitteiston sydän. Ohjausyksikön muistiin tallennetaan muun muassa laiteohjelmisto sekä sovellusohjelmat yksikön älykästä toimintaa varten. Tulo- ja lähtöportit muodostavat ohjausyksikön rajapinnan ulkomaailmaan, mikä mahdollistaa tiedon vaihdon ja toiminnan ympäristön kanssa. Ilman prosessorin tarjoamia toimintoja ohjaavaa laiteohjelmistoa, ilman ennalta ohjelmoituja loogisia operaatioita sisältävää sovellusohjelmaa ja ilman tulo- ja lähtöporttien tarjoamaa rajapintaa kommunikaatioon ulkomaailman kanssa, ei ohjausyksikön älykäs toiminta olisi mahdollista. [1]



Kuva 1: Älykkään ohjausyksikön perusarkkitehtuuri.

2.2. Lähtökohdat ja tavoitteet

Tämän työn tavoite on tuottaa testausongelmaan ratkaisu, jolla saavutetaan riittävän hyvin testaukselle asetetut kriteerit. Testaukselle asetettujen kriteerien saavuttamiseksi on laiteohjelmiston toiminta voitava varmentaa oikeelliseksi kaikkien sen sovellusohjelmoijalle tarjoamien prosessoriin, muistin käsittelyyn ja kommunikaatioon liittyvien ominaisuuksien osalta. Testauksen riittävän hyvän laadun ja toistettavuuden saavuttamiseksi tarkoituksena on rakentaa testausjärjestelmä, jolla voidaan suorittaa kattavia testejä ohjausyksiköiden laiteohjelmiston testaamiseksi niin, että laiteohjelmiston laatu vastaa luotettavasti sille asetettuja vaatimuksia.

2.3. Vaatimukset

Tärkeimpänä vaatimuksena testausjärjestelmälle on sen modulaarisuus. Testausjärjestelmän tulee olla helposti laajennettavissa niin laitteiden osalta kuin myös ohjelmistollisesti. Modulaarisuus on tärkein vaatimus, koska järjestelmän elinkaaren määrää hyvin pitkälle se, kuinka helposti ja nopeasti sitä voidaan laajentaa ja muokata uusien ohjausyksiköiden testaukseen sopivaksi tulevaisuudessa. Testausjärjestelmällä tulee voida suorittaa erilaisten ohjausyksiköiden testaus automaattisesti ilman, että järjestelmää itsessään täytyisi muuttaa erityisesti, vaan muutosten tulisi kohdistua ohjelmiston kautta valittaviin parametreihin, jotka määrittävät erilaiset toimintatilat erilaisten ohjausyksiköiden tapauksissa. Jokainen testauksen kohteena oleva ohjausyksikkö sisältää samanlaisen laiteohjelmiston, mutta ohjausyksikön I/O-rajapinnat vaihtelevat yksikkökohtaisesti.

Toinen keskeinen vaatimus testausjärjestelmälle on sen automaattisuus. Testien automatisoinnilla saadaan aikaan testitapausten helppo suoritus sekä toistettavuus. Samankaltaisia testejä tulee voida toistaa useita kertoja useille erilaisille ohjausyksiköille, jolloin voidaan varmistua laiteohjelmiston toimintojen oikeellisuudesta erilaisissa ohjausyksiköissä. Automatisoinnin myötä kokonaistestausaika lyhenee huomattavasti, koska järjestelmällä voidaan suorittaa nopeasti monimutkaisia testitapauksia, joihin manuaalisella testauksella kuluisi aikaa huomattavasti enemmän. Manuaalinen testaus on aikaa vievää ja saattaa oikein suoritettuna kestää jopa useita viikkoja, kun automaattisella testausjärjestelmällä tavoitellaan testausaikoja, joita mitataan tunneissa tai jopa minuuteissa riippuen testien toistojen määrästä. Automaattinen testausjärjestelmä voi suorittaa testejä helposti ympäri vuorokauden tarpeen niin vaatiessa, joten laiteohjelmistoa voidaan testata tavoilla, joita ei ole ennen ollut mahdollista toteuttaa.

2.4. Rajaukset

Tämä työ sisältää automaattisen testausjärjestelmän toteutuksen määrittelyvaiheesta ohjelmointiin sisältäen myös järjestelmän toiminnallisen testauksen. Laitteiston käyttöohjeet sekä dokumentaatio eivät muuten sisälly varsinaiseen diplomityön alueeseen, vaan ne to-

teutetaan diplomityöstä erillisenä osana automaattitesteri-projektissa, johon diplomityön aihe kuuluu.

Diplomityö käsittää vain testausjärjestelmän puoleisen toteutuksen, joten testien kohteina olevien ohjausyksiköiden ohjelmointia ei käsitellä tässä työssä. Ohjausyksiköiden automaattinen testaus vaatii yksikkökohtaista ohjelmointia myös testattavan ohjausyksikön puolella. Tämä ohjelmointi suoritetaan projektin puitteissa, mutta se ei sisälly diplomityön alueeseen.

3. Älykkäät koneenohjausjärjestelmät

Tässä luvussa esitellään hajautetun koneenohjausjärjestelmän arkkitehtuuri ja toiminta pääperiaatteissaan. Koneenohjausjärjestelmän tärkeimpänä yksittäisenä osana käydään läpi älykkään ohjausyksikön toiminta, joka on perusta koko koneenohjausjärjestelmälle. Ohjausyksikön toiminnalle tärkeimpinä pääkohtina esitellään sen arkkitehtuuri, I/O-liitynnät sekä kommunikaatiorajapinnat.

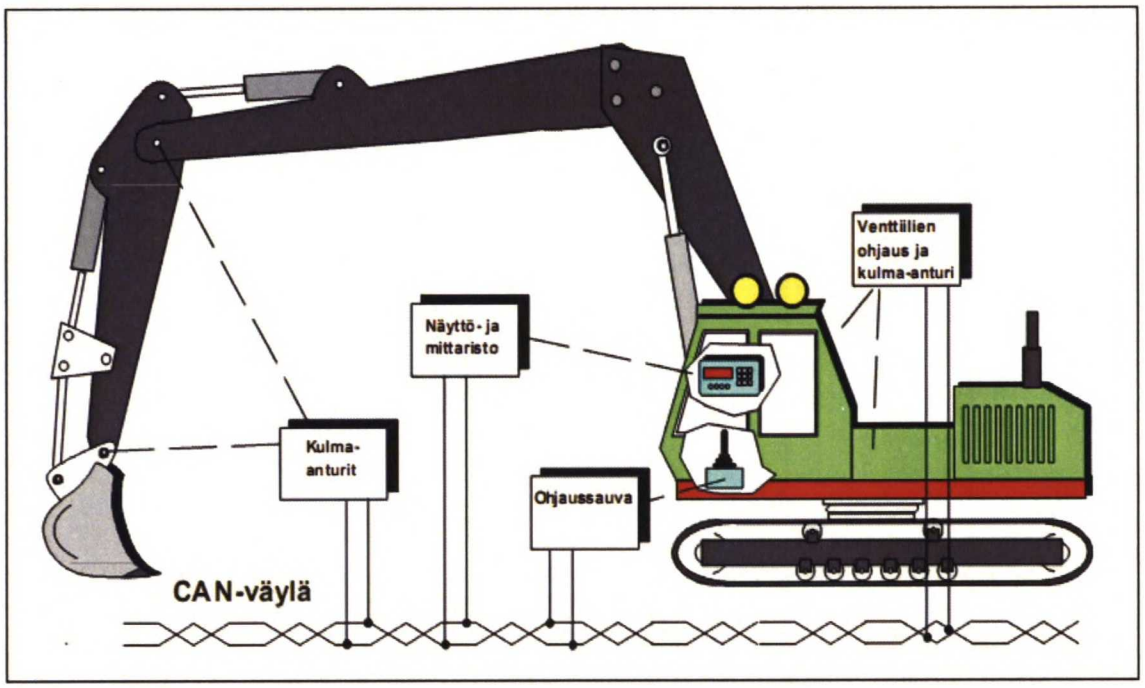
3.1. Hajautettu ohjausjärjestelmä

Hajautetulla ohjausjärjestelmällä tarkoitetaan järjestelmää, johon kuuluu enemmän kuin yksi prosessointiyksikkö eli ohjausyksikkö. Ohjausyksiköt vaihtavat tietoa keskenään järjestelmässä jonkinlaisen tietoliikennejärjestelmän välityksellä. Hajautetun ohjausjärjestelmän tarkoitus on toteuttaa käytön aikaiset toiminnallisuudet esimerkiksi työkoneessa. [2]

Järjestelmän kukin osa hoitaa omaa itsenäistä tehtäväänsä osana järjestelmäkokonaisuutta, myös silloin kun jokin järjestelmän osa vikaantuu tai on muuten tilapäisesti poissa käytöstä. Tämä parantaa olennaisesti järjestelmän käytettävyyttä, luotettavuutta ja turvallisuutta, koska riski koko järjestelmän lamautumisesta pienentyy olennaisesti hajautuksen ansiosta.

Hajautetun älyn keskeisin ominaisuus on, että tietoja voidaan käsitellä prosessorissa ja varastoida muistiin lähellä niiden käyttö- ja tuotantopaikkaa, jolloin raskasta tiedonsiirtoa saadaan olennaisesti vähennettyä. Ohjausjärjestelmän eri osat voidaan sijoittaa helposti pienen kokonsa ansiosta lähelle käyttöpaikkaansa, mikä helpottaa laitteiden ja koneiden huoltoa sekä tekee antureiden ja toimilaitteiden kytkemisen ohjausyksikköön vaivattomaksi. Kääntöpuolena ohjausyksiköiden sijoittaminen lähelle käyttöpaikkaansa aiheuttaa muun muassa haasteita lämpötilan, kosteuden ja tärinän kanssa, minkä seurauksena ohjausyksiköiden pitää olla erityisesti suunniteltu kestämään vaativia olosuhteita. [3]

Kuvassa 2 on esitetty kaivinkoneen hajautettu ohjausjärjestelmä. Järjestelmään on merkitty paikat, joissa voisi olla sijoitettuna ohjausyksikkö. Kaivinkoneen hajautetussa järjestelmässä, tässä kuvitteellisessa tapauksessa, on sijoitettu erillinen ohjausyksikkö ohjaamaan kurottajan nivelien kulma-antureita, hytin näyttöä ja mittaristoa, kaivinkoneen ohjaussauvaa sekä koneen sisäisiä venttiilejä ja kulma-antureita. Tässä tapauksessa kommunikaatioväylänä toimii CAN-väylä, jota ohjausyksiköt käyttävät keskinäiseen kommunikointiin. Ohjausyksiköt voivat jakaa väylässä muun muassa antureiden tuottamia arvoja, joita voidaan sitten käyttää hyväksi muissa koneen osissa, kuten esimerkiksi hytin mittariston osoittimissa. [4]



Kuva 2: Kaivinkoneen hajautettu ohjausjärjestelmä. [4]

3.2. Älykäs ohjausyksikkö

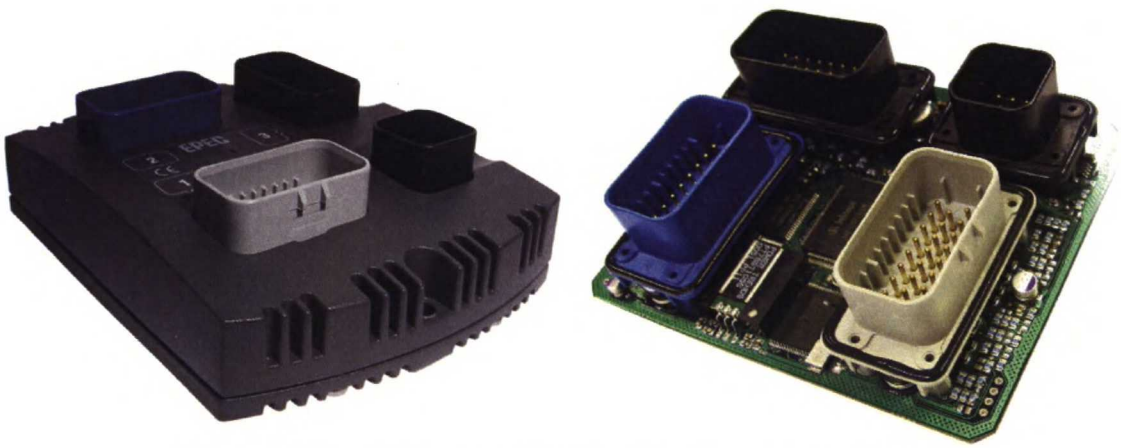
Ohjausjärjestelmän perusmoduuleina eli ohjausyksikköinä voivat olla esimerkiksi ohjelmoitavat logiikat (PLC), PC-koneet sekä erilaiset mikroprosessoreihin perustuvat ohjausmoduulit yleisesti. [2]

Tärkeimpänä yhdistävänä tekijänä kaikissa älykkäissä ohjausyksiköissä on mikroprosessori, joka mahdollistaa monimutkaiset laskentaoperaatiot ja tätä kautta ohjelmoinnin. Ohjausyksikkö koostuu yksinkertaisimmillaan tulo- ja lähtömoduuleista tai integroiduista tuloista ja lähdöistä, mikroprosessorista ja muisteista. Ohjausyksikön liityntä ohjausjärjestelmään tapahtuu tulojen ja lähtöjen välityksellä yleensä jonkin standardin kommunikaatiorajapinnan, kuten CAN-väylän, kautta. Lisäksi tuloihin ja lähtöihin voidaan kytkeä erilaisia antureita ja toimilaitteita. [1]

Ohjausyksikön laitteiston päälle sijoittuu heti laiteohjelmisto, joka toimii rajapintana laitteiston eli fyysisen piirikytkennän ja ylemmän tason sovellusten välillä. Laiteohjelmisto ohjaa muun muassa prosessoria, tuloja ja lähtöjä sekä muistien toimintaa.

Kuvassa 3 on esitetty tämän työn kannalta varsin olennainen ohjausyksikkö Epec 2024. Epec 2024 on yksi automaattisen testausjärjestelmän testauksen kohteena olevista ohjausyksiköistä. Hankaliin ympäristöolosuhteisiin tarkoitettu, vesi- ja pölytiivis, ohjausyksikkö

sisältää muun muassa Infineonin 40 MHz C167 prosessorin ja 512 kilotavua RAM-muistia sekä 1 Mb Flash-muistia. +24 voltin käyttöjännitteellä toimivasta ohjausyksiköstä löytyy myös kaksi toisistaan erillistä CAN-liityntää, digitaalituloja, digitaalilähtöjä, analogiatuloja, pulssituloja sekä PWM-lähtöjä. [3]



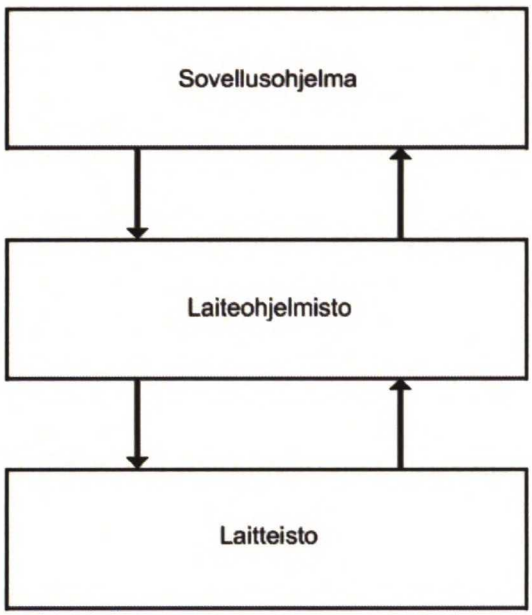
Kuva 3: Epec 2024 yleiskäyttöinen koneenohjausyksikkö.

3.2.1. Arkkitehtuuri

Kuten edellä on jo mainittu, älykäs ohjausyksikkö koostuu yksinkertaisimmillaan prosessorista, muisteista ja tulo- sekä lähtömoduuleista. Käytännössä tällaisia yksinkertaisia ohjausyksiköitä ovat ohjelmoitavat logiikat. Jos esimerkiksi PC-konetta käytetään ohjausyksikkönä, voi yksikkö käsittää periaatteessa monia erilaisia lisälaitteita, kuten esimerkiksi näytönohjaimen.

Kuvassa 1 sivulla 3 on jo esitetty älykkään ohjausyksikön perusarkkitehtuurin lohkokaavio. Arkkitehtuuri vastaa lähimmin ohjelmoitavan logiikan sisäistä laitteiston rakennetta, mutta toimii perustana kaikille ohjausyksiköille. Älykäs ohjausyksikkö on siis mikroprosessoriin pohjautuva laite, jossa on joko modulaarisia tai integroituja lähtö- ja tuloportteja. Lähtö- ja tuloportteihin kytketään kentällä olevia antureita ja toimilaitteita. Ohjausyksikkö ohjaa toimilaitteita sisäiseen muistiinsa tallennetun sovellusohjelman ja antureiden tuottamien signaalien mukaisesti. [5]

Kuvassa 4 on esitetty tässä työssä testattavien älykkäiden ohjausyksiköiden ohjelmistoarkkitehtuuri. Ohjausyksikön fyysisen laitteiston päälle sijoittuu laiteohjelmisto, jossa on toteutettu ohjelmallisesti laitteiston tarjoamien toimintojen ohjaus. Laiteohjelmiston yläpuolella on ohjelmistoarkkitehtuurissa sovellusohjelma, jossa on toteutettu ohjausyksikön suorittamat älykkäät toiminnot, kuten esimerkiksi erilaiset ohjausalgoritmit ja tietojenkäsittely. Sovellusohjelma käyttää laitteiston toimintoja laiteohjelmiston tarjoamien kirjastofunktioiden kautta eli laiteohjelmisto toimii rajapintana fyysisen laitteiston ja sovelluksen välissä.



Kuva 4: Älykkään ohjausyksikön ohjelmistoarkkitehtuuri.

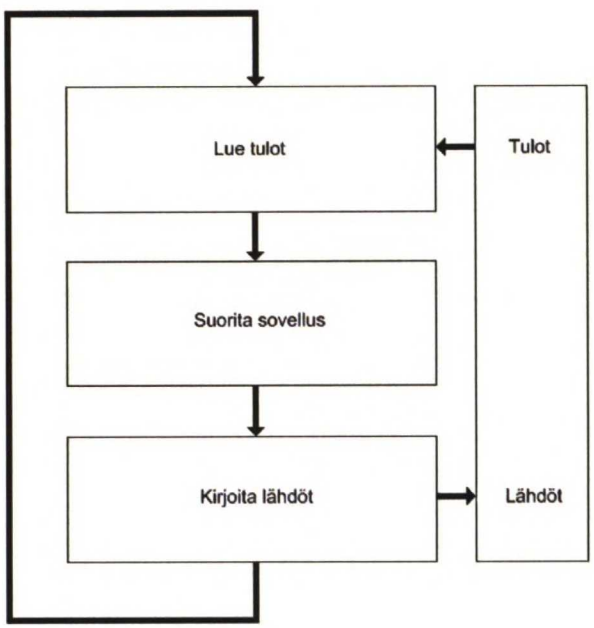
3.2.2. Laiteohjelmisto

Laiteohjelmisto on laitteen haihtumattomaan muistiin, esimerkiksi flash-muistiin, tallennettu ohjelmisto, joka toimii eräänlaisena rajapintana itse laitteiston (hardware) ja sovellusohjelmiston (software) välissä kuvan 4 mukaisesti. Laiteohjelmiston tunnetumpi nimitys ammattikielessä on firmware, joka on vakiintunut käytettäväksi yleisesti myös suomen kielessä. Se ohjaa laitteiston tarjoamia toimintoja ja tarjoaa näin ollen sovellusohjelmistolle rajapinnan laitteiston ohjaamiseen. Sovellusohjelmoija voi ohjata laitteiston toimintoja ja näin ollen toteuttaa ohjausyksikön älykkyyden käyttämällä laiteohjelmiston tarjoamia kirjastofunktioita. Kirjastofunktiot tarjoavat sovellusohjelmoijalle pääsyn laitteiston kaikkiin laiteohjelmistossa määriteltyihin toimintoihin. [6]

Tässä työssä testattavien ohjausyksiköiden laiteohjelmisto tarjoaa muun muassa CAN-väylän yläpuolelle toteutetun ylemmän tason kommunikaatioprotokollan, CANopenin, palvelut sovellusohjelmoijan käyttöön. Laiteohjelmistossa on myös toteutettu ohjausyksikön tulojen ja lähtöjen ohjaus. Muun muassa ohjausyksikön analogiatulojen mitattava analoginen suure, virta tai jännite, valitaan laiteohjelmiston avulla. Laiteohjelmisto asettaa myös prosessorin PWM-lähtöjen taajuuden ja ohjaussuhteen sovellusohjelmoijan valitsemaksi ja alustaa ohjausyksikön pulssitulot toimimaan halutulla tavalla. Kaikki laiteohjelmiston ominaisuudet ovat sovellusohjelmoijan käytössä kirjastofunktioiden välityksellä, jolloin sovellusohjelmoijan ei tarvitse ottaa kantaa kirjasto funktioiden sisäiseen toteutukseen. [7]

Laiteohjelmisto voi sijaita valmiiksi ajettavassa muodossa ohjausyksikön haihtumattomassa muistissa ja se voidaan suorittaa sieltä ilman alustustoimenpiteitä. Toinen mahdollisuus on, että laiteohjelmisto sijaitsee ohjausyksikön haihtumattomassa muistissa ja se alustetaan ohjausyksikön käynnistyksessä suoritettavaan muotoon ohjausyksikön työmuistiin, joka voi olla esimerkiksi RAM-muisti. Yksinkertaisimmillaan laiteohjelmisto muodostaa yksinään koko laitteen ohjelmiston. Monimutkaisemmissa ohjausyksiköissä laiteohjelmisto saattaa kuitenkin toteuttaa ainoastaan alemman tason toimintoja. Tällaisia alemman tason toimintoja voivat olla esimerkiksi itse suoritettavan sovellusohjelmiston lataus työmuistiin tai käyttöjärjestelmän käynnistyskoodin aktivointi käynnistyksessä. Laiteohjelmiston suorittaessa ainoastaan alatason toimintoja, monimutkaisemmat sovellusohjelmat hoitavat vaativammat älykkäät toiminnot käyttämällä hyväksi laiteohjelmistossa toteutettuja funktioita. [6]

Kuvassa 5 on esitetty laiteohjelmiston kiertoperiaatteellinen toiminta. Käytännössä tämä tarkoittaa sitä, että laiteohjelmiston ollessa käynnissä se pyörii silmukassa suorittaen toimintoja järjestyksessä "Lue tulot" – "Suorita sovellus" – "Kirjoita lähdöt". Toiminnon "Lue tulot" yhteydessä laiteohjelmisto lukee ohjausyksikön tulojen arvot ja päivittää ne muistiinsa. Tämän jälkeen laiteohjelmisto suorittaa sovellusohjelman. Kierroksen lopuksi laiteohjelmisto kirjoittaa muistissaan olevat lähtöjen arvot ohjausyksikön lähtöihin, minkä jälkeen ohjelmakierros alkaa alusta.



Kuva 5: Laiteohjelmiston kiertoperiaatteellinen toiminta.

Tässä työssä testauksen kohteena olevissa älykkäissä ohjausyksiköissä laiteohjelmisto sijaitsee ohjausyksikön flash–muistissa, josta sitä ajetaan suoraan ohjausyksikön käynnistyk-

sen jälkeen. Automaattisella laiteohjelmiston testausjärjestelmällä on tarkoitus testata laiteohjelmiston toiminnallisuutta ja ominaisuuksia.

3.2.3. Sovellusohjelmointi

Sovellusohjelma sijoittuu älykkään ohjausyksikön ohjelmistoarkkitehtuurissa laiteohjelmiston yläpuolelle sivun 9 kuvan 4 esittämällä tavalla. Sovellusohjelma tallennetaan laiteohjelmiston tavoin ohjausyksikön haihtumattomaan muistiin, esimerkiksi flash-muistiin, mutta se ladataan käyttöä varten ohjelmamuistiin (RAM) toisin kuin laiteohjelmiston tapauksessa. Sovellusohjelmaa ei siis suoriteta suoraan haihtumattomasta muistista. Sovellusohjelmointi tarjoaa laitteen ohjelmoijalle pääsyn ohjausyksikön laiteohjelmiston tarjoamiin laitteiston ominaisuuksiin ja tätä kautta mahdollisuuden käyttää niitä erilaisten älykkäiden toiminnallisuuksien luontiin. Sovellusohjelman ja laiteohjelmiston ero on, että laiteohjelmisto sijaitsee hierarkkisesti lähempänä fyysistä laitteistoa kuin sovellus. Laiteohjelmisto ei toteuta ohjausyksikön älykästä toimintaa, toisin kuin sovellusohjelma, vaan se ohjaa ainoastaan laitteiston toimintoja sovellusohjelman vaatimalla tavalla.

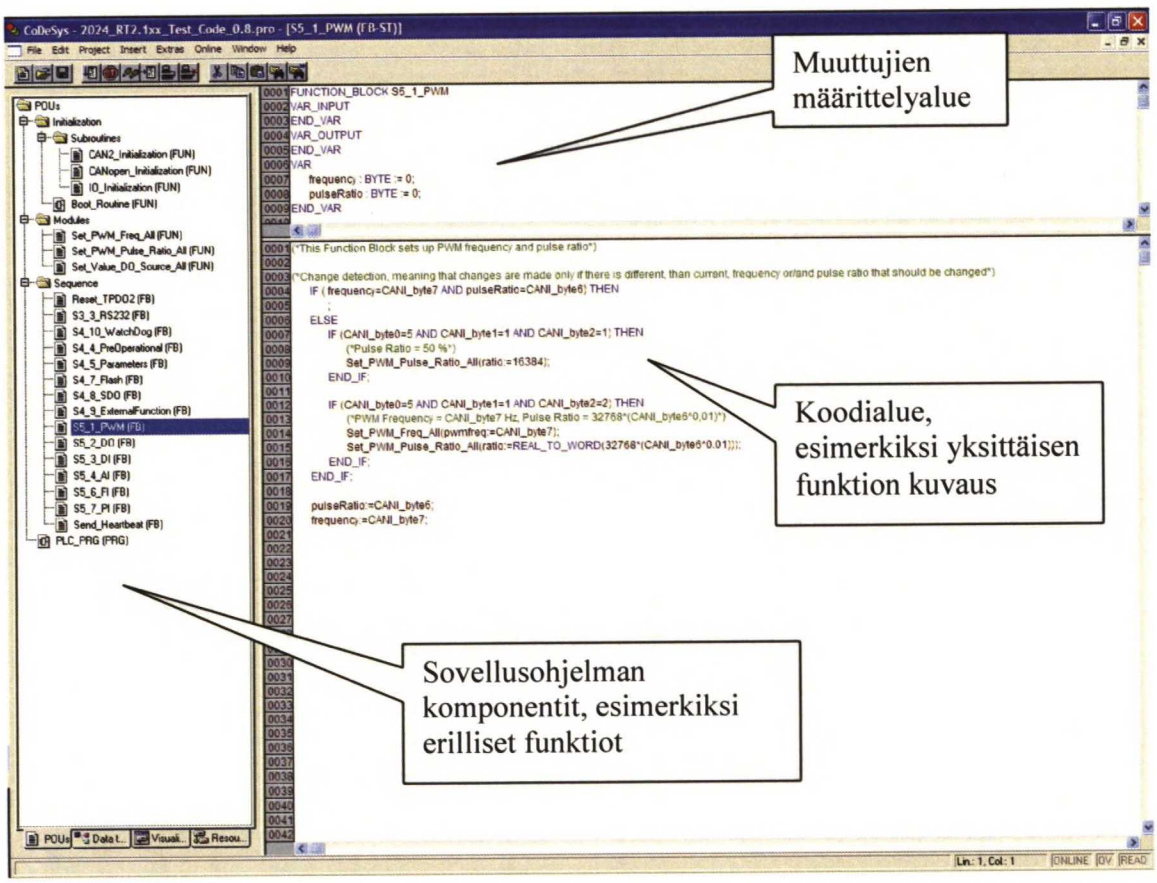
Tämän työn kohteena olevan automaattisen laiteohjelmiston testausjärjestelmän testattavien ohjausyksiköiden sovellusohjelmointi perustuu avoimeen IEC 61131-standardiin. Standardi on tarkoituksellisesti kehitetty yhteneväksi ja avoimeksi rungoksi ohjelmoitavien logiikoiden arkkitehtuurille [8]. Standardi on jaettu kahdeksaan eri osaan, jotka ovat:

- Osa 1: Yleinen informaatio
  - Määrittelee standardin sisällön ja sovelluskohteet.
- Osa 2: Laitevaatimukset ja testit
  - Määrittelee lukuisia erilaisia laitteistovaatimuksia ohjelmoitaville logiikoille ja niihin liittyville oheislaitteille sekä asettaa ohjelmoitavien logiikoiden toiminnallisuuteen, turvallisuuteen ja testaukseen liittyviä vaatimuksia.
- Osa 3: Ohjelmointikielet
  - Määrittelee syntaksin (lauseoppi) ja semantiikan (merkitysoppi) ryhmälle ohjelmoitaville ohjausyksiköille tarkoitettuja ohjelmointikieliä.
- Osa 4: Käyttäjän ohjeet
  - Esittelee loppukäyttäjille standardin sekä auttaa heitä määrittelemään ja valitsemaan laitteistovaatimuksensa standardin mukaisesti.
- Osa 5: Kommunikaatio
  - Kuvaa kommunikaatiopalvelut ohjelmoitavien ohjausyksiköiden väliseen kommunikaatioon ohjelmoijan näkökulmasta.
- Osa 6: Kenttäväyläkommunikaatio
  - Osa on poistettu vuonna 2000. Odottaa kenttäväylästandardin IEC 61158 osien 5 ja 6 valmistumista.
- Osa 7: Sumea ohjaus
  - Sumean logiikan ohjelmointi osan 3 (IEC 61131-3) määrittelemillä kielillä.

- Osa 8: Ohjeita ohjelmointikielien käyttöönottoon ja implementointiin
  - Ohjeita sovelluskehittäjille ohjelmointiin osan 3 (IEC 61131-3) määrittelemillä kielillä.

IEC 61131-3 on kolmas osa avoimesta kansainvälisestä IEC 61131-standardista [9]. Se käsittelee ja määrittelee ohjelmoitavien ohjausyksiköiden ohjelmointikieliä, joita on kaksi tekstipohjaista ja kolme graafista. Standardin määrittelemiä ohjelmointikieliä ovat: tekstipohjaiset IL (Instruction List) ja ST (Structured Text) sekä graafiset LD (Ladder Diagram) ja FBD (Function Block Diagram). Standardi määrittelee myös sovelluksen sisäiseen organisointiin tarkoitetun graafisen SFC (Sequential Function Chart) -kielen. Tämä ryhmä ohjelmointikieliä on tarkoitettu määritelmänsä mukaan ohjelmoitavien ohjausyksiköiden ohjelmointiin. Lisäksi standardin kolmannessa osassa on määritelty joukko konfigurointielementtejä, jotka on tarkoitettu tukemaan sovellusohjelman asennusta laitteistoon, sekä ominaisuuksia, jotka helpottavat ohjausyksiköiden kommunikaatiota automaatiojärjestelmän muiden osien kanssa. [10]

Tässä työssä IEC 61131-3-standardin mukaisten sovellusten kehitykseen ohjausyksiköille käytetään kuvassa 6 esitettyä CoDeSys-sovelluskehitysympäristöä (Controller Development System). Sillä on mahdollista luoda sovelluksia kaikilla viidellä IEC 61131-3 standardin mukaisella ohjelmointikielellä ja se sisältää kokonaisvaltaiset työkalut ohjelmoitavien ohjausyksiköiden sovelluskehityksen tarpeisiin. Kehitysympäristö sisältää sovelluskehitystyökalujen lisäksi muun muassa valmiit toiminnot sovelluksen kääntämiseen ja sen asennukseen ohjausyksikköön sekä reaaliaikaisen muuttujien seurannan ohjausyksikön ollessa toiminnassa. [11]



Kuva 6: CoDeSys-sovelluskehitysympäristö.

3.3. Ohjausyksikön I/O-liitynnät

Älykkään ohjausyksikön tärkeimpiä ominaisuuksia on sen kytkentärajapinta ulkomaailmaan. Ilman tuloihin (Input) kytkettyjä antureita ja lähtöihin (Output) kytkettyjä toimilaitteita ohjausyksikkö ei voi olla vuorovaikutuksessa ympäristönsä kanssa. Ohjausyksikkö saa erilaisten anturien, kuten esimerkiksi lämpötila-anturien ja kytkimien, kautta informaatiota toimintaympäristöstään sisääntuloihinsa. Tuloista ohjausyksikkö saa prosessoitavan informaationsa, jonka se muuttaa laitteistonsa ja sovelluksen erilaisten algoritmien avulla vasteeksi. Vaste konkretisoituu toimilaitteissa, kuten esimerkiksi venttiileissä ja releissä, joita ohjausyksikkö ohjaa lähtöjensä (Output) avulla. [12]

Reaktiivisen ohjausyksikön toimintaa voidaankin verrata ihmisaivoihin, jotka prosessoivat informaatiota. Ihminen tekee havaintoja esimerkiksi silmiensä ja kuulonsa avulla, jotka vastaavat ohjausyksikön antureita. Aivot prosessoivat ilmaisimien synnyttämän informaation sovellusohjelman tapaan, minkä jälkeen aivot tuottavat vasteen toimilaitteille, joita ovat ihmisen tapauksessa kädet ja jalat. Tämä yksinkertainen esimerkki osoittaa havainnol-

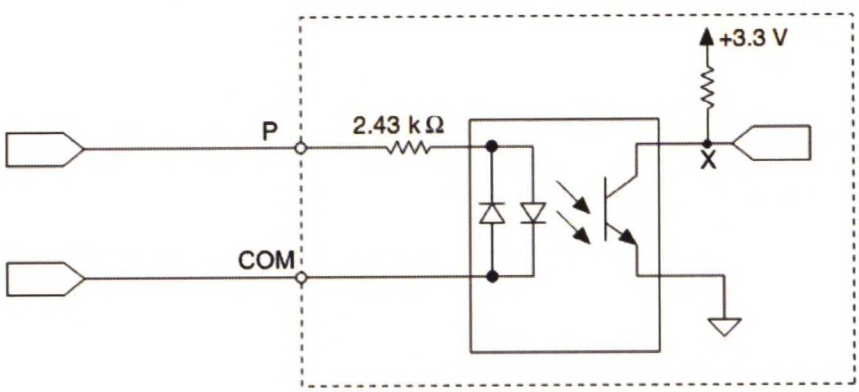
lisesti kuinka tärkeä rajapinta ulkomaailmaan on älykkäälle ohjausyksikölle. Ilman I/O-rajapintaa ei ohjausyksiköllä voida suorittaa hyödyllisiä älykkäitä toimintoja. [12]

3.3.1. Digitaaliset liitynnät

Digitaalisella tuloliitynnällä on kaksi loogista tilaa: päällä tai pois. Ohjausyksikön kannalta digitaalisella tulolla ei voi olla muita tiloja kuin edellä mainitut. Tämä tarkoittaa sitä, että esimerkiksi digitaalinen paikka-anturi ei tuota informaatiota siitä, kuinka lähellä tai kaukana ollaan jostain paikasta, vaan anturi ilmaisee ainoastaan asetettujen rajojen puitteissa sen, onko jokin raja ylitetty vai ei (päällä/pois). Tämä ei kuitenkaan tarkoita sitä, että digitaaliseen tuloon ei voisi olla kytkettynä myös analogista signaalia, esimerkiksi jännitettä, tuottava anturi. Tällaisessa tapauksessa ohjausyksikön puolella on määritelty jänniterajat sille, milloin tulo on päällä ja milloin pois.

Yleisiä ohjausyksikön digitaaliseen tuloliityntään kytkettäviä ilmaisimia ovat muun muassa erilaiset painonapit, rajakatkaisijat, digitaaliset paikka-anturit ja valosilmät. [13]

Kuvassa 7 on esitetty esimerkki digitaalisen tuloliitynnän mahdollisesta kytkennästä. Kytkennässä tulolinjaan P kytketään positiivinen jännite tai nollajännite ja linjaan COM negatiivinen jännite tai nollajännite. Linja COM voi olla myös kytkettynä nollajännitteeseen eli maapotentiaaliin ohjausmoduulin sisällä. Kytkentä toimii siten että, kun linjaan P kytketään nollaa suurempi positiivinen jännite, niin valodiodi alkaa johtaa ja kytkee transistorin johtavaan tilaan, jolloin virta pääsee kulkemaan transistorin läpi ja piste X kytkeytyy maapotentiaaliin. Transistorin ollessa johtamattomassa tilassa pisteen X potentiaali on kuvan kytkennän tapauksessa +3,3 volttia. Transistorin ohjauksen toteutus tehdään yleensä kuvan tapaan valodiodin avulla, jolloin piiriin syntyy optoerotus, joka suojaa herkkiä logiikkapiirejä, kuten prosessoria, ylijännitteiltä ja ylivirroilta.



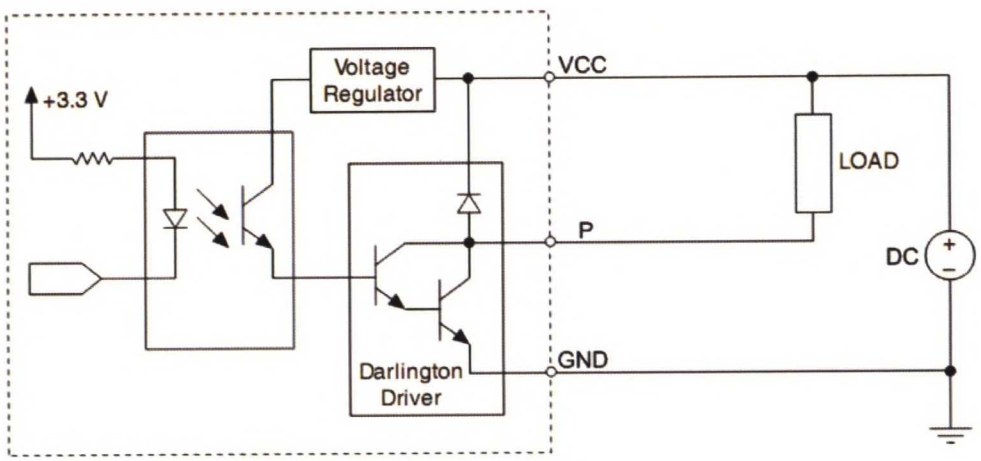
Kuva 7: Digitaalinen tuloliityntä. [14]

Digitaalinen lähtöliityntä toimii käänteisesti verrattuna digitaaliseen tuloliityntään siinä mielessä, että loogisen tilan (päällä/pois) valinta suoritetaan ohjausyksikössä, tarkemmin sanottuna sovellusohjelmassa. Esimerkiksi +24 V-järjestelmässä digitaalisen lähdön tilat voivat olla aseteltuina siten, että kun lähtö on asetettu päälle se antaa +24 V jännitteen ja kun lähtö on asetettu pois se antaa 0 V jännitteen. Myös käänteinen logiikka on mahdollinen.

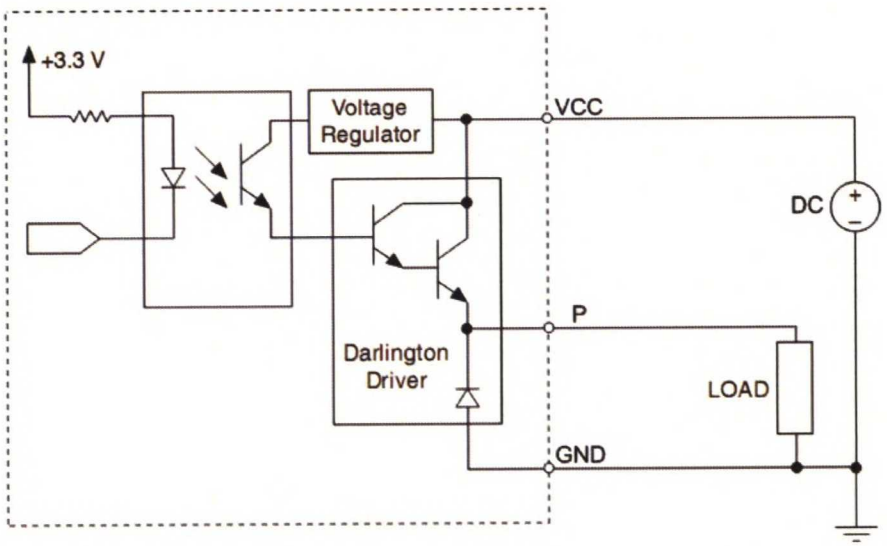
Yleisiä digitaalisiin lähtöihin kytkettäviä toimilaitteita ovat releet, käynnistysmoottorien solenoidit ja erilaiset venttiilit. [13]

Kuvissa 8 ja 9 on esitetty kaksi mahdollista tapaa toteuttaa digitaalinen lähtöliityntä. Kytkentä kuvassa 8 toimii siten, että kun halutaan antaa ohjausta lähtöliitynnälle, kytketään valodiodi johtavaksi, jolloin valodiodin ohjaama transistori alkaa johtaa antaen ohjaussignaalin darlington-kytkennälle (Darlington Driver). Darlington-kytkennän saadessa ohjaussignaalin, se alkaa johtaa, jolloin kuorman (LOAD) ollessa kytkettynä kuvan mukaisesti, kuorma kytkeytyy linjan P kautta maapotentiaaliin eli linjaan GND kytkettyyn potentiaaliin. Darlington-kytkennän saadessa ohjaussignaalia, se on johtavassa tilassa ja kuorman yli vaikuttaa linjaan VCC kytketty jännite.

Digitaalisen lähtöliitynnän kytkennän ollessa kuvan 9 kaltainen, kytkentä toimii muuten samaan tapaan kuin kuvan 8 kytkentä, mutta darlington-kytkennän saadessa ohjaussignaalia kuorma kytkeytyy linjan P kautta linjaan VCC kytkettyyn potentiaaliin maapotentiaalin sijaan.



Kuva 8: Maapotentiaaliin kytkevä digitaalinen lähtöliityntä. [14]



Kuva 9: Jännitteeseen kytkevä digitaalinen lähtöliityntä. [14]

3.3.2. Analogiset liitynnät

Analogiset tuloliitynnät ottavat vastaan jatkuvia suureita kuten jännite- ja virta-arvoja. Toisin kuin digitaaliset tulot, analogiset tulot voivat ottaa vastaan anturilta minkä tahansa jatkuvan arvon niiden määrittelyalueen sisältä. Esimerkiksi analoginen jännitetulo, jolle on määritelty toiminta-alue 0–5 V voi ottaa vastaan minkä tahansa arvon kyseiseltä määrittely-väliltä lukutarkkuuden puitteissa. Analogia-arvot muutetaan ohjausyksikön ymmärtämään digitaaliseen muotoon analogia-digitaali-muuntimella ohjausyksikön sisällä. Yleisiä analogisia tulosuureita ovat esimerkiksi nopeus, lämpötila ja paine, joita mitataan erilaisilla analogisilla antureilla.

Analoginen lähtöliityntä tuottaa, joko jatkuvan jännite- tai virtasuureen. Analogisen lähdön arvo määrätään ohjausyksikön sovelluksessa ja se on määrättävissä vapaasti tarkkuuden ja toiminta-alueen rajoissa. Sovellusohjelmoinnissa analogialähdölle annetaan digitaalinen arvo, joka muutetaan digitaali-analogia-muuntimella analogiseksi suureeksi ohjausyksikön sisällä. Analogisella lähdöllä voidaan ohjata esimerkiksi moottorin kierroslukua, painetta ja erilaisten venttiilien asentoa. [13]

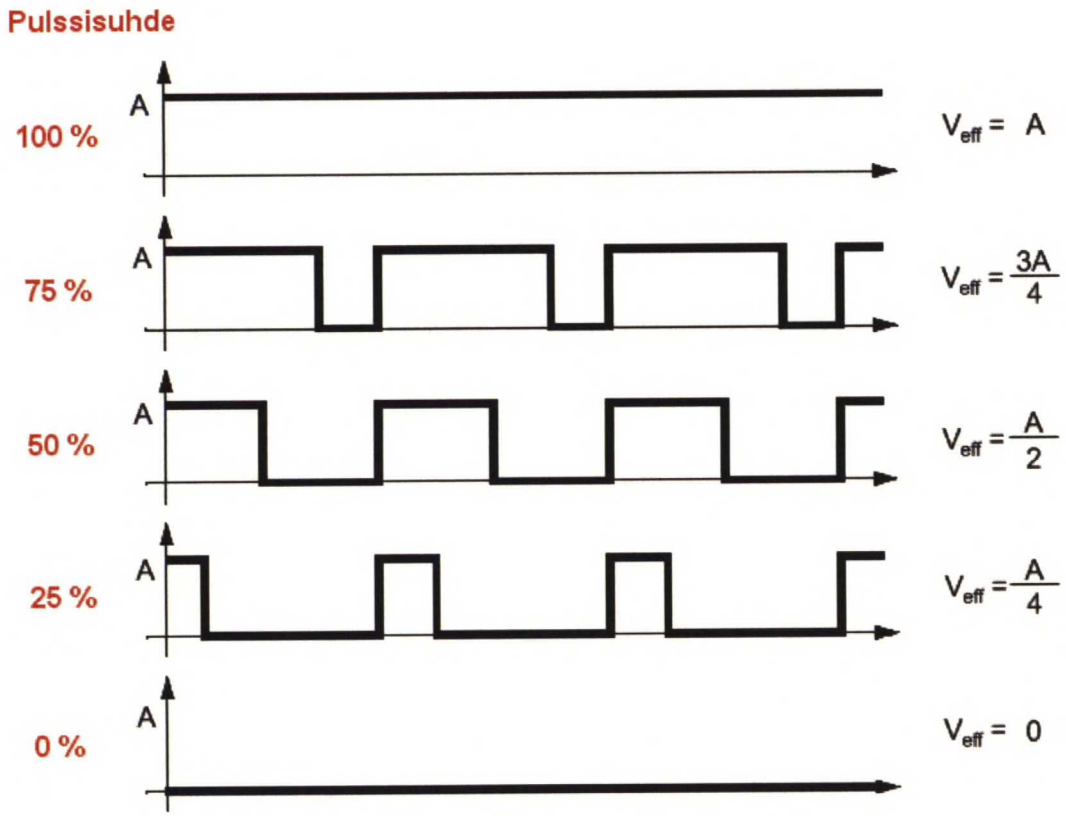
3.3.3. Pulssinleveysmodulointi

Analoginen lähtöjännite voidaan tuottaa myös käyttämällä pulssinleveysmodulointia kuvan 10 esittämällä tavalla. Kuvassa esitettyä pulssinleveysmodulointitapaa kutsutaan myös digitaaliseksi pulssinleveysmodulaatioksi, koska lähtöjännite voi saada diskreetit arvot A (päällä), joka on jännitteen maksimiarvo, ja 0 V (poissa), joka on jännitteen miniarvo [15]. Pulssinleveysmodulaation avulla saadaan muodostettua kanttiaalto käyttämällä tietyn ajan

maksimiarvoa ja tietyn ajan miniarvoa eli toisin sanoen lähtöjännite on välillä päällä ja välillä poissa määrätyn ajan. Lähtöjännitteeksi muodostuvan kanttiaallon taajuus on korkea, tyypillisesti korkeampi kuin kuuloalueen yläraja, joka on 20 kHz. Kanttiaallon pulssisuhde määrää efektiivisen lähtöjännitteen eli lähtöjännitteen keskiarvon. Pulssisuhde tarkoittaa jännitteen maksimiarvon päälläoloajan suhdetta minimiarvon päälläoloaikaan prosentteina. Esimerkiksi 100 % pulssisuhde tarkoittaa, että maksimiarvo on päällä koko ajan. 0 % pulssisuhde tarkoittaa taasen, että jännite on koko ajan minimiarvossaan. Jos pulssisuhde on esimerkiksi 50 %, käytetään jännitteen maksimi- ja minimiarvoa yhtä kauan. [8]

Pulssinleveysmodulointia käytetään yleisesti tehoelektroniikassa. Pulssinleveysmoduloitua signaalia voidaan käyttää esimerkiksi servomoottorin tehonsyöttönä. Servomoottoria syötettäessä lähtöjännitteen korkeat taajuudet suodattuvat automaattisesti pois, johtuen servomoottorin resistanssien ja induktanssien aiheuttamasta pitkästä aikavakiosta (alipäästösuodatus). Muissa tapauksissa, joissa luonnollinen aikavakio ei ole tarpeeksi pitkä, voidaan käyttää myös alipäästösuodattimia suodattamaan pulssinleveysmoduloidusta signaalista pois korkeat taajuudet analogisen tasajännitesignaalin aikaansaamiseksi. [8]

Pulssinleveysmodulaatiota on perusteltua käyttää monissa eri tapauksissa. Pulssinleveysmodulaatiossa kytkimenä toimiva elektroninen komponentti on suurimman osan ajasta joko johtavassa tilassa tai estotilassa, jolloin virran kulku on estetty kokonaan. Tästä johtuen komponentissa ei tapahdu suurta tehohäviötä ja hyötysuhde pysyy korkeana verrattuna muuttuvaan resistanssiin perustuviin säätötapoihin, joissa osa tehosta kuluu resistiivisen komponentin häviöihin. Pulssinleveysmodulaatio on myös helppo, nopea ja tarkkuusvaatimuksista riippuen, myös mahdollisesti halpa tapa tuottaa ohjausyksiköstä analoginen lähtöjännite. Tarkkuusvaatimuksien kasvaessa myös pulssinleveysmodulaatiolla tuotetun jännitteen hinta kasvaa, koska muun muassa jännitteen tuottamiseen vaadittujen elektronisten komponenttien hinta kasvaa niiltä vaaditun tarkkuuden kasvaessa. [15]



Kuva 10: Pulssinleveysmoduloituja signaaleja. [8]

3.4. Kommunikaatio

Älykkään hajautetun koneenohjausjärjestelmän tietoliikenteen toteuttamiseksi on tarjolla lukuisia erilaisia vaihtoehtoja. Moniprosessorijärjestelmissä, todella pienillä etäisyyksillä, voidaan ja yleensä kannattaakin turvautua jonkin rinnakkaismuotoisen prosessoriväylän käyttöön. Etäisyyksien kasvaessa useisiin metreihin kannattaa yleensä siirtyä käyttämään sarjamuotoista kenttäväylää tietoliikenteeseen, koska sarjamuotoisen tiedonsiirron ansiosta sähkömagneettisten häiriöiden hallinta on paljon helpompaa etäisyyksien kasvaessa kuin rinnakkaismuotoisessa tiedonsiirrossa. Sarjamuotoinen kommunikaatio onkin yleensä aina käytössä, kun siirretään tietoja fyysisesti erillisten ohjausyksiköiden välillä. Kenttäväylä on yleisnimitys sarjamuotoiselle tiedonsiirtoväylälle, jota käytetään I/O:n ja prosessoinnin hajautuksessa. Kenttäväylään liittyy aina sekä fyysinen sarjaliikenneyhteys että tiedonsiirtoprotokolla. Kenttäväylä sallii älykkyyden ja toimintojen hajautuksen suurenkin koneen alueella.

Asynkroninen kommunikaatio hajautetussa järjestelmässä on myös yksi hyvin yleisesti käytetty ratkaisu tietoliikenteen toteuttamiseen, varsinkin jos solmujen lukumäärä ei järjestelmässä ole kovin suuri. Asynkroniset kommunikaatioratkaisut ovat yleensä niin sanottuja pisteestä pisteeseen yhteyksiä, jolloin solmujen välinen kommunikointi ei tapahdu yhden yhteisen väylän kautta kuten esimerkiksi kenttäväylissä, vaan kommunikaatio tapahtuu erikseen kahden pisteen välillä pisteestä pisteeseen. [2]

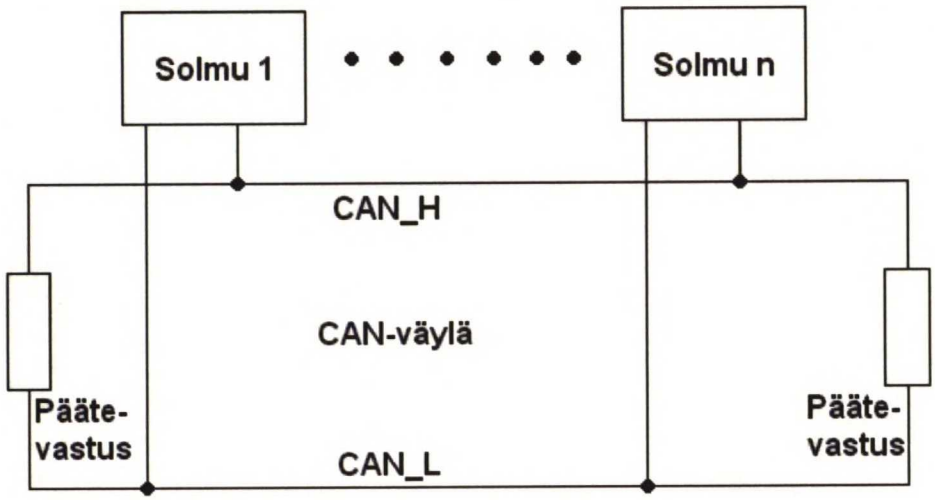
Seuraavaksi esitellään tämän työn kannalta olennaiset tietoliikennöintitavat hajautetussa ohjausjärjestelmässä. Ensimmäiseksi käydään läpi sarjamuotoiseen kenttäväylään perustuva ratkaisu CAN sekä CAN-väylän päälle rakennettu ylemmän tason protokolla CANopen. Tämän jälkeen esitellään hyvin yleinen asynkroninen pisteestä pisteeseen kommunikointitapa RS-232. Lopuksi käydään läpi fyysinen sarjamuotoinen kommunikaatioratkaisu RS-485.

3.4.1. CAN – "Controller Area Network"

CAN-väylä suunniteltiin alkujaan vuonna 1986 autojen hajautettujen ohjausjärjestelmien reaaliaikaisen tiedonsiirron tarpeisiin. CAN on siis luonteeltaan ajoneuvoväylä, mutta se on nykyään saanut paljon jalansijaa myös työkoneiden ja teollisuuden erilaisissa sovelluksissa. Suomalaisen Kone Oy:n hissisovellukset olivat yksi ensimmäisiä CAN-väylän teollisuussovelluksia maailmassa 1980-luvun lopulla. CAN on tiedonsiirtoväylä, joka sopii periaatteessa mihin tahansa koneeseen, jossa käytetään lyhyitä tiedonsiirtoyhteyksiä ja lyhyitä viestejä. CAN-väylää käytetään usein kenttäväylän tapaan anturi- ja toimilaiteväylänä. CAN ei periaatteeltaan ole kuitenkaan kenttäväylä, vaan ennemminkin koneen sisäinen prosessoriverkko, jonka tehtävänä on tuoda eri solmujen tilanmuuttujat toisilleen näkyviksi, jotta järjestelmän rinnakkainen ohjaus olisi mahdollista. [4]

CAN on moni-isäntäinen järjestelmä, jonka väylätopologia on linjamainen ja toimintatapa reaaliaikainen [16]. Se on lisäksi digitaalinen sarjaliikenneväylä, joka yltää aina 1 Mbit/s tiedonsiirtonopeuksiin lyhyillä siirtoetäisyyksillä (< 25 metriä). Siirtoetäisyyksien kasvaessa myös väylän nopeutta on pudotettava. 10 kbit/s nopeuksilla väylä voi olla jopa 5 kilometriä pitkä. [17]

CAN-väylä muodostuu yksinkertaisimmillaan kahdesta kierretystä johtimesta, joissa kulkee CAN_H- (CAN_High) ja CAN_L (CAN_Low) -signaalit, kuten kuvassa 11 on esitetty. Kaikki CAN-väylään liitetyt solmut, eli tämän työn tapauksessa ohjausyksiköt, yhdistetään johdolla, jonka päihin laitetaan päätevastukset (yleensä n. 120 Ω). Yleensä väylä on niin pitkä, että päätevastukset on hyvä laittaa väylän molempiin päihin, mutta vaatimuksena on, että ainakin toisessa päässä täytyy olla päätevastus. Päätevastuksen tarkoitus on estää heijastumat väylän päistä.



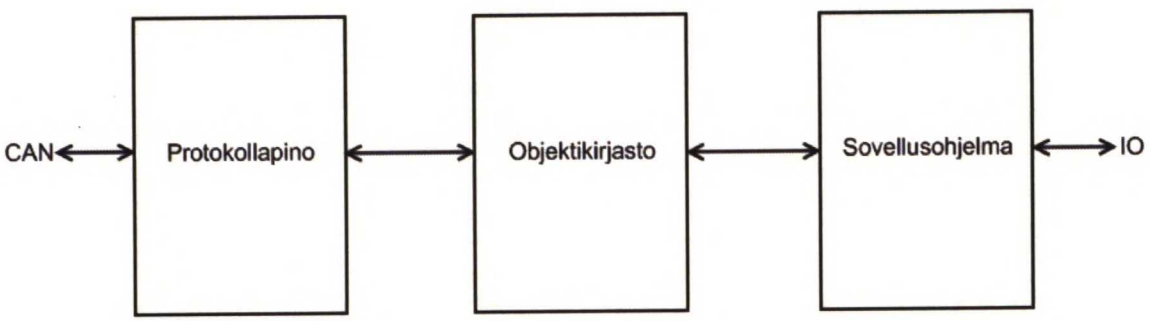
Kuva 11: CAN-väylän sähköinen kytkentä.

Tässä työssä testattavien ohjausyksiköiden ensisijaisena ja yleensä ainoana tietoliikenneväylänä toimii CAN-väylä. Ohjausyksiköt käyttävät CAN-väylää kommunikointiin ohjausjärjestelmissä, joissa on useita ohjausyksiköitä suorittamassa paikallisia toimintoja. Ohjausjärjestelmän ohjausyksiköiden täytyy jakaa tilanmuuttujia keskenään, jotta hajautettu ohjaus olisi mahdollista.

3.4.2. CANopen

Fyysisen CAN-väylän päälle on rakennettu myös useita korkeamman tason protokollia, joilla on pyritty varmistamaan eri laitevalmistajille mahdollisuus tuoda markkinoille keskenään vaihdettavissa olevia standardoituja komponentteja. Yksi näistä korkeamman tason protokollista on CANopen, johon seuraavaksi keskitytään enemmän, koska tämän työn testauslaitteiston testikohteena olevat ohjausyksiköt toteuttavat CANopen-protokollan.

CANopen-standardin mukainen laite koostuu kolmesta loogisesta osasta, joilla laite liitetään toiselta puolelta sovelluksen I/O-dataan ja toiselta puolelta CAN-väylään. Kuvan 12 CANopen-laitemalli sisältää protokollapinon, joka toimii kommunikaatiorajapintana CANopen-laitteen kytkemiseksi CAN-väylään. Objektikirjasto toimii sovellusohjelman ja CAN-väylän välisenä rajapintana. Sovellusohjelma hoitaa laitteen kytkennän I/O-dataan ja toteuttavaa laitteen halutut toiminnot. [16]



Kuva 12: CANopen-protokollan laitemalli. [16]

CANopen-standardin keskeinen elementti on objektikirjasto, jossa laitteen toiminnallisuus, parametrit ja muuttujat ovat kuvattuina. Taulukossa 1 on kuvattuna objektikirjaston sisältö. Objektikirjaston maksimi koko on 65536 arvoa, joita osoitetaan 16-bittisen indeksin avulla. Lisäksi jokaiselle indeksille on määritelty 8-bittinen alaindeksi, joten kukin indeksi voi sisältää 256 alaindeksiä. Osa objektikirjaston objekteista on kaikille CANopen-laitteille yhteisiä ja osa kuuluu vain tiettyyn laiteprofiiliin. CANopen-standardi määrittelee useita erilaisia laiteprofiileja, kuten esimerkiksi I/O-moduulit ja käytöt, jättäen kuitenkin paljon liikkumavaraa valmistajakohtaisille määrittelyille. Jokaiselle laiteprofiilille on standardissa määritelty tietyt pakolliset objektikirjaston arvot. Tällä on voitu varmistaa, että tiettyyn laiteprofiiliin kuuluvat laitteet ovat toiminnallisesti samanlaisia ja yhteensopivia. [17]

Taulukko 1: Objekti kirjaston rakenne. [18]

Indeksi (Hex)	Objektin tyyppi
0000	Ei käytössä
0001-001F	Staattiset datatyypit
0020-003F	Monimutkaiset datatyypit
0040-005F	Valmistajakohtaiset monimutkaiset datatyypit
0060-007F	Laiteprofiilikohtaiset staattiset datatyypit
0080-009F	Laiteprofiilikohtaiset monimutkaiset datatyypit
00A0-0FFF	Varattu tulevaan käyttöön
1000-1FFF	Kommunikaatioprofiilin alue
2000-5FFF	Valmistajakohtaisen profiilin alue
6000-9FFF	Standardoidun laiteprofiilin alue
A000-BFFF	Standardoidun rajapintaprofiilin alue
C000-FFFF	Varattu tulevaan käyttöön

CANopenin fyysinen kerros, CAN-väylä, kykenee lähettämään ainoastaan lyhyitä paketteja, jotka sisältävät 11-bittisen ID:n, RTR-bitin (Remote Transmission Request), joka kertoo onko kyseessä normaali dataviesti vai datan lähetyspyyntö, sekä 0-8 tavua dataa. 11-bittinen CAN-kehyksen ID jakautuu standardin mukaan vielä 4-bittiseen funktiokoodiin ja 7-bittiseen CANopen-solmunumeroon. CANopenissa CAN-kehyksen ID:tä kutsutaan COB-ID:ksi (Communication Object Identifier). Tämä tarkoittaa sitä että CANopen-laitteita voidaan kytkeä enintään 127 samaan väylään, kun solmunumero 0 on kielletty. CAN-väylän laajennus, CAN 2.0 B, määrittelee myös 29-bittisen ID-kentän, jolloin samaan väylään voidaan liittää useampia laitteita. Tämä laajennus ei kuitenkaan ole kovin usein tarpeellinen verkkojen koosta johtuen. Taulukossa 2 on esitetty standardin mukainen CANopen-kehys. [19]

Taulukko 2: Standardin CANopen-kehyksen sisältö. [19]

	Funktiokoodi	Solmunumero	RTR	Datan pituus	Data
Pituus	4 bittiä	7 bittiä	1 bitti	4 bittiä	0-8 tavua

CANopenissa käytetään erilaisia kommunikaatiomalleja viestien lähetykseen CANopen-solmujen välillä. Näitä kommunikaatiomalleja on kolme: [18]

- Isäntä/Orja-suhde:
  - Yksi CANopen-solmu on määritetty isännäksi, joka lähettää tai pyytää dataa orjasolmuilta. Isäntäsolmu hallinnoi väylän liikennettä.

- Asiakas/Palvelin-suhde:
  - Asiakassolmu pyytää objektikirjaston tietoja lähettämällä SDO-kyselyn palvelimelle. Palvelinsolmu vastaa asiakassolmulle lähettämällä pyydetyt tiedot.

- Tuottaja/Kuluttaja-suhde - Työntö/Veto-malli:
  - Työntömallissa tuottaja lähettää tietoa kuluttajalle ilman, että kuluttajan täytyy erikseen tietoja pyytää. Vetomallissa kuluttaja pyytää tuottajalta ensin tietoja, minkä jälkeen tuottaja lähettää pyydetyt tiedot kuluttajalle.

Laitemallissa kuvassa 12 CAN-väylän puolella oleva CANopen-protokollapino koostuu erilaisista tiedonsiirtoon tarvittavista protokollista, joiden avulla väylällä olevia CANopen-laitteita hallitaan: [18]

- PDO-protokolla (Process Data Object)
  - PDO-prokollan avulla voidaan siirtää sovellustietoja verkon laitteiden välillä. Sovellustiedot voivat olla esimerkiksi muuttujien tai I/O-liityntöjen arvoja.

- SDO-protokolla (Service Data Object)
  - SDO-protokollaa käytetään verkossa olevien CANopen-laitteiden objektikirjastojen arvojen lukemiseen ja kirjoittamiseen. Objektikirjastoon voi olla tallennettuna esimerkiksi tietyn laitteen I/O-liityntöjen arvot, joita toiset verkon laitteet voivat lukea SDO-viestien avulla Asiakas/Palvelin-suhteen mukaisesti.

- NMT-protokolla (Network Management)
  - NMT-protokollalla hallitaan CANopen-laitteiden toimintaa. NMT-viesteillä voidaan muun muassa siirtää verkossa olevat laitteet tiloihin: Operational, Stop ja Pre-Operational, jotka ovat CANopen-laitteen erilaisia toimintatiloja. NMT-protokolla toimii Isäntä/Orja-suhteen mukaisesti.

- SYNC-protokolla
  - SYNC-protokolla synkronoi verkon toimintaa. Synkronointi tapahtuu siten että SYNC-tuottajaksi valittu laite lähettää verkkoon periodisesti SYNC-viestejä, joita SYNC-kuluttajat vastaanottavat. Nämä SYNC-objektit toimivat verkon sisäisenä synkronointikellona.

- EMCY-protokolla
  - Ilmaisee laitteen sisäisiä vikoja. Kun laitteessa tapahtuu sisäinen vikatilanne, laite lähettää tuottajana EMCY-viestin verkkoon, jolloin muut kuluttajina toimivat laitteet voivat reagoida halutulla tavalla.

- TimeStamp-protokolla
  - Asettaa verkkoon yhtenäisen ajan. TimeStamp-tuottaja lähettää verkkoon TimeStamp-viestin, josta kuluttaja laitteet saavat yhteisen ajan.

- Error Control-protokolla
  - Käytetään virhetilanteiden havainnointiin ja korjaamiseen. Voidaan toteuttaa kahdella eri tavalla. Node Guarding-protokollassa isäntä-laite valvoo kyselyviestien avulla, että orja-laite on toimintakunnossa, jos orja-laite ei vastaa kyselyyn, laitteessa on vika. Toimii myös siten että, kun isäntä-laite ei lähetä kyselyä orja-laitteelle, orja-laite tietää, että isäntä-laitteessa on vika. Heartbeat-protokollassa heartbeat-tuottaja lähettää heartbeat-viestiä periodisesti verkkoon, jolloin heartbeat-kuluttaja huomaa tuottajan vikatilanteet heartbeat-viestien loppuessa.

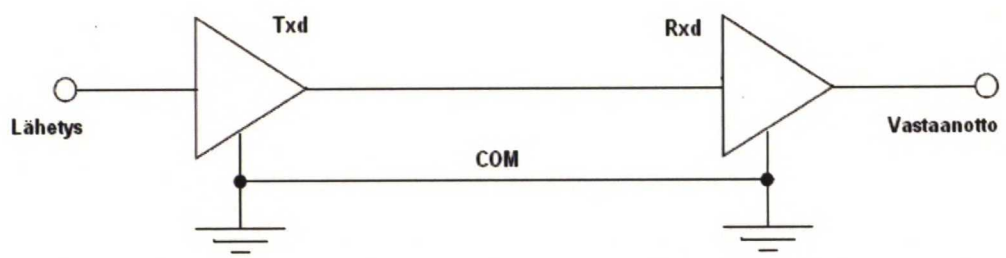
Kuvan 12 laitemallissa sovellusohjelma on CANopen-laitteita valmistavien yritysten tärkein osaamisalue. Sovellusohjelma toteuttaa laitteen objektikirjaston liitynnän I/O-dataan sekä toteuttaa laitteen älykkäät toiminnot. Tässä työssä testauksen kohteena olevien ohjausyksiköiden sovellusohjelmointi toteutetaan IEC 61131-standardin mukaisesti kohdassa 3.2.3 Sovellusohjelmointi esitettyyn tapaan.

3.4.3. RS-232-sarjaväylä

RS-232 on standardi, joka määrittelee kahden tietokonelaitteen välisen sarjamuotoisen datasignaalikommunikaation. Standardi määrittelee muun muassa datasignaalien sähköiset ominaisuudet, kuten jännitetasot, sekä kommunikaatiorajapinnan mekaaniset ja funktionaaliset ominaisuudet. Standardi ei ole kuitenkaan kaikenkattava ja näin ollen siinä ei oteta kantaa esimerkiksi merkkien koodaukseen, lähetyksen bittinopeuksiin, jotka ovat asynkronisuudesta johtuen yleensä melko alhaisia, tai siirtovirheen etsintään. [20]

Sarjamuotoisessa kommunikaatiossa kommunikointiväylälle lähetetään yksi bitti kerrallaan, minkä ansiosta tietoliikennöintiin tarvitaan ainoastaan yksi kommunikointikanava. Tämän ansiosta kustannukset pysyvät matalina, mutta varjopuolena on että ainoastaan yksi käytettävissä oleva kanava rajoittaa saavutettavaa maksimi kommunikointinopeutta. Kommunikaatiokanavat toteutetaan yleensä sähköjohdoilla, mutta myös radiosignaalien tai optisten kaapelien käyttö on mahdollista.

RS-232 on yleisin käytössä oleva standardi, joka perustuu jännitetasojen muutoksiin. Tämä tarkoittaa käytännössä sitä, että lähettävän tietokonelaitteen päässä väylälle lähtevän bitin arvo voi olla joko tosi (1) tai epätosi (0). Kanavaohjain muuttaa loogisen arvon lähettävän laitteen päässä Txd-jännitearvoksi, joka voi saada arvon väliltä epätosi: +3 – +15 V ja tosi: -3 – -15 V. Lähettävän laitteen lähettävä- eli Txd-linja ja maataso eli COM kytketään kaapelilla vastaanottavan laitteen vastaanotto- eli Rxd-linjaan ja COM:iin. Vastaanottava laite muuntaa lopulta positiiviset ja negatiiviset jännitearvot takaisin loogisiksi arvoiksi, jolloin ne voidaan tulkata bittiarvoiksi. Kuvassa 13 on esitettynä RS-232-standardin määrittelemä sähköinen kytkentä edellä olevan selityksen mukaan. Laitteisto voi toimia myös käänteisesti eli siten, että lähettäjä toimii vastaanottajana ja vastaanottaja lähettäjänä, jolloin yksinkertaisimmillaan molemmista tietokonelaitteistoista löytyy ainakin kolme liityntälinjaa eli linjat Rxd, Txd ja COM. [8]



Kuva 13: RS-232-standardin mukainen sähköinen väyläkytkentä. [8]

Kuvassa 13 lähetyspään lähtevä signaali on Txd (Transmitted Data). Vastaanottopäässä tuleva signaali on Rxd (Received Data) ja COM on yhteinen maataso lähettimelle ja vastaanottimelle.

Edellä mainittu kolmilinjainen kaksisuuntainen tietoliikennöinti on yksinkertaisin mahdollinen versio RS-232-standardin mukaisesta kommunikaatiosta. Standardi kuitenkin määrittelee myös monia muita signaaleja kommunikaation toteutuksen tueksi, jolloin kommunikaatiosta voidaan toteuttaa suuremmalla linjamäärällä parempi ja luotettavampi versio. [20]

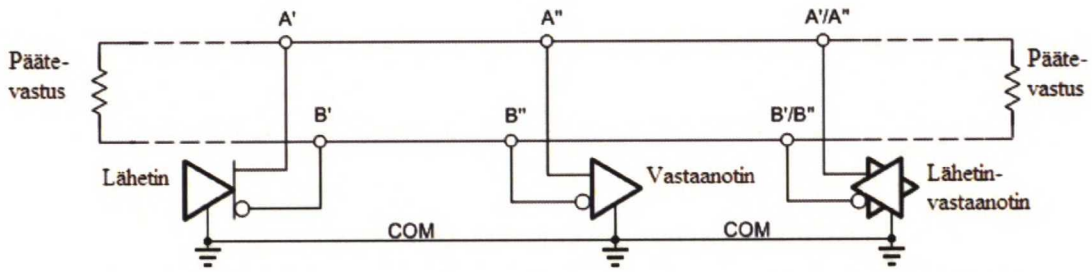
RS-232-standardin versio C julkaistiin jo vuonna 1969, minkä jälkeen sen nimitys on muuttunut moneen otteeseen sitä sponsoroivan organisaation nimen muuttuessa. Standardiin on julkaistu myös erilaisia pienempiä päivityksiä, mutta uudempien versioiden mukaan toteutetut laitteistot ovat yhteensopivia aiemmin julkaistujen versioiden mukaan toteutettujen kanssa. Tällä hetkellä standardin uusin versio on nimeltään TIA-232-F. [20]

Tämän työn testauksen kohteena olevista ohjausyksiköistä löytyy RS-232-liityntä, jonka käyttötarkoitus on jätetty loppukäyttäjän määriteltäväksi. Tämä tarkoittaa, että ohjausyksiköistä löytyy RS-232-liityntä sekä kommunikaatiorajapinta. Sovellusohjelmoija voi käyttää valmista ohjelmointikirjastoa toteuttaakseen RS-232-kommunikaation jonkin toisen laitteen kanssa. Ohjausyksikkö voi siis periaatteessa kommunikoida minkä tahansa ulkoisen laitteen kanssa, mistä löytyy RS-232-liityntä.

3.4.4. RS-485-differentiaalinen sarjaväylä

RS-485 on standardi, joka määrittelee differentiaalisen sarjaväylän, johon voidaan liittää useita väylälaitteita samanaikaisesti. RS-485-standardi tunnetaan nykyään nimellä TIA-485-A, mutta vakiintunut standardin alkuperäinen nimitys RS-485 on vielä laajasti käytössä.

RS-485-standardi määrittelee ainoastaan kommunikointiväylän sähköiset ominaisuudet eli väylän fyysisen kerroksen ottamatta kantaa esimerkiksi tiedonsiirtoprotokolliin tai käytettäviin liittimiin. Kolmijohtimisessa RS-485-väylässä tietoliikenne tapahtuu puolittain kaksisuuntaisesti (half-duplex). Tämä tarkoittaa sitä, että ainoastaan yksi väylällä oleva laite voi lähettää väylälle tietyllä ajan hetkellä. RS-485-väylä voidaan toteuttaa myös viisijohtimisena, jolloin lähettäminen ja vastaanottaminen onnistuvat yhtä aikaa (full-duplex). Koska väylä on differentiaalinen signaloinniltaan ja toteutetaan yleensä kierretyllä parikaapelilla, se on yhteismuotoisten häiriöiden kumoutumisen takia paljon häiriönsietokykyisempi kuin RS-232-väylä. Tämän takia RS-485-väylä sopii paremmin käytettäväksi pitkille matkoille (< 1200 m; nopeudella 100 kbit/s) ja häiriöisiin ympäristöihin. [21]



Kuva 14: RS-485-standardin mukainen sähköinen väyläkytkentä. [8]

Kuvassa 14 A', A'' ja A'/A'' ovat ei-invertoivia pinnejä ja B', B'' ja B'/B'' ovat invertoivia pinnejä. Lähetin, vastaanotin ja lähetin-vastaanotin ovat väylään kytkettyjä laitteita, jotka voivat osallistua kommunikaatioon väylällä. Väylä on lisäksi terminoitu päätevastuksilla, mikä on suositeltava tapa hoitaa kytkentä, mutta myös toimilaitteen asentaminen väylän päähän on mahdollista. Päätevastuksien tarkoitus on minimoida signaalien heijastukset väylän päistä.

RS-485-väylässä käytetään usein isäntä-orja-järjestelyä, mikä tarkoittaa, että yksi väylällä olevista laitteista toimii isäntänä, joka ohjaa väylän liikennettä samaan tapaan kuin kenttäväylissä voidaan toimia. Väylää voidaan kuitenkin käyttää myös pisteestä pisteeseen kommunikointiin RS-232-väylän tavoin. [21]

Tämän työn testauksen kohteena olevissa ohjausyksiköissä RS-485-kommunikaation käyttökohde on, RS-232-kommunikaation tavoin, jätetty vapaasti loppukäyttäjän määriteltäväksi.

4. Automaattinen testaus

Automaattinen testaus on nykyään kasvava trendi niin ohjelmistopuolella kuin myös erilaisten laitteiden toiminnan verifioimisessa. Laitteiden ja ohjelmistojen monimutkaistuessa ja niiden koon kasvaessa on päädytty tilanteeseen, missä automaattinen testaus on ainoa järkevä ratkaisu testauksen toteuttamiseksi niin, että järjestelmien ja ohjelmistojen toiminnallisuuksien verifiointi on laadullisesti ja ajallisesti järkevällä tasolla. Automaattinen testaus asettaa omat vaatimuksensa testauslaitteistolle ja sen ohjelmoinnille verrattaessa tavalliseen, ei automatisoituun, testauslaitteistoon ja sen ohjelmointiin. Luvussa käydään läpi tämän työn automaattisen laiteohjelmiston testausjärjestelmän kannalta olennaista testauksen teoriaa ja erityispiirteitä. Luvussa käsitellään myös järjestelmän kannalta keskeisiä erityispiirteitä testauslaitteistolle sekä laitteiston ohjelmoinnin toteutukseen tarvittavat ohjelmistot sekä tekniikat.

4.1. Testaus

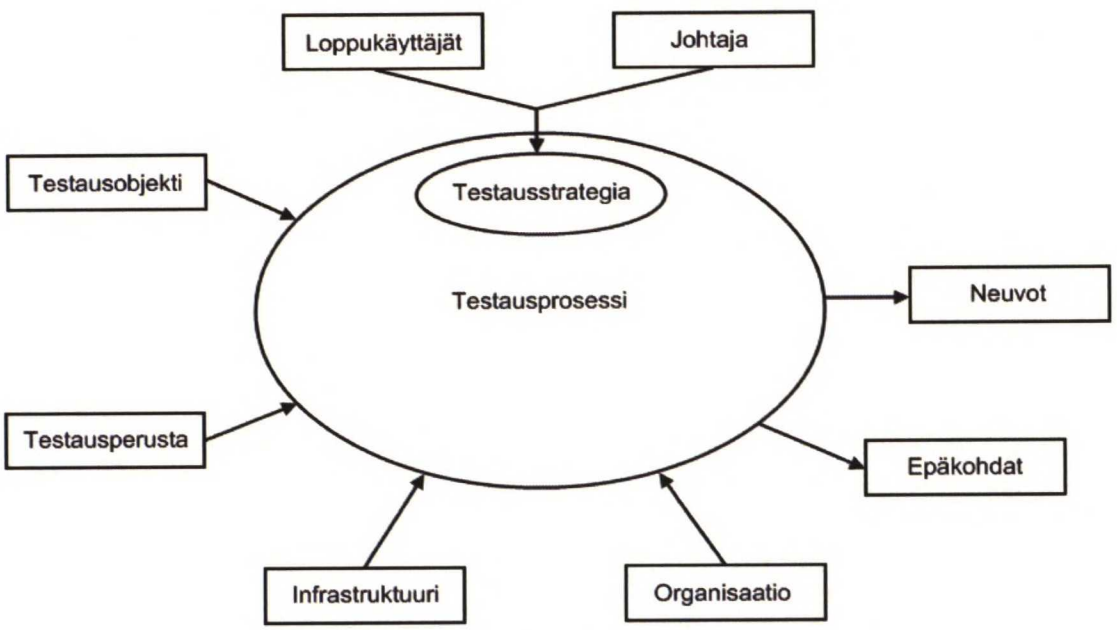
Testaus on prosessi, joka pyrkii löytämään jonkin järjestelmän epäkohtia. Testausta voidaan suorittaa niin virheiden poistamisen tarkoituksessa kuin myös hyväksynnän näkökohdista. Virheitä poistamalla pyritään ohjaamaan järjestelmä tilaan, jossa se toimii, kuten on määritelty. Hyväksyntänäkökohdista testausta suoritetaan, kun halutaan varmentaa järjestelmän oikeellinen toiminta ennen sen julkaisua tai vastaavaa tilasiirtymää. Epäkohtien löytäminen järjestelmistä on kuitenkin joka tapauksessa testauksen tärkein tarkoitus. Vaikka onkin varsin selvää, että on parempi estää epäkohtien syntyä jo ennen prosessin testausvaihetta, niin käytännössä ei kuitenkaan ole mahdollista tuottaa systeemejä, jotka olisivat vapaita virheistä. Testaus onkin siis pysyvä ja hyvin tärkeä osa kaikkien järjestelmien kehitysprosessia – se auttaa tuottamaan laadukkaampia järjestelmiä.

Testaus ei suoraan paranna järjestelmien laatua, vaan sen on tarkoitus ohjata järjestelmien kehitysprosessia oikeaan suuntaan, antamalla arvokkaita neuvoja ja tietoja. Testauksella tuotettujen tietojen avulla kehitysprosessin osalliset voivat tehdä päätöksiä, joiden kautta järjestelmän laatua voidaan kehittää oikeaan suuntaan. Testauksen tuottamia tietoja ovat tiedot järjestelmien epäkohdista, joita ovat muun muassa järjestelmien puutteet, virheet ja toiminnalliset ristiriidat.

Jokainen testausprosessi tarvitsee suunnitelman testauksen tarpeista. Suunnitteluprosessin aikana täytyy tehdä tarkat määrittelyt siitä, mitä järjestelmästä tulisi testata ja miten testit tulisi suorittaa. Kartoittamalla testauksen tarpeet voidaan lähteä kehittämään itse testausprosessia, joka täyttää vaatimusten määrittelemät testien yksityiskohdat. Testiprosessia suunniteltaessa täytyy muistaa, että kaikkia järjestelmän epäkohtia on mahdoton löytää ja kaikkia järjestelmän toiminnallisuuksia ei ole mahdollista testata. Testausprosessin määrittelyssä onkin ensisijaisen tärkeää suunnitella ja kohdentaa testausresurssien käyttö viisaasti.

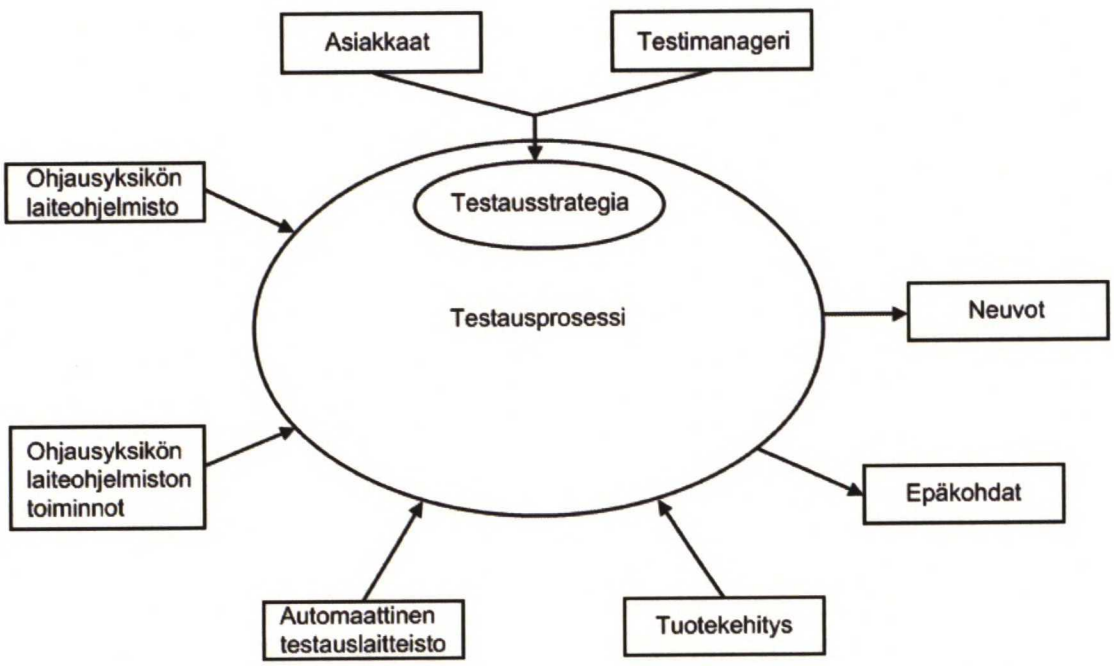
Testattavien toiminnallisuuksien määrää on helppo laajentaa automaattisen testauksen avulla, minkä pohjalta tätäkin diplomityötä on osaltaan lähdetty tekemään.

Kuvassa 15 on esitetty yleinen testausprosessi, jota voidaan soveltaa yleisien elementtiensä kautta jokaiseen hyvin organisoituun testausprosessiin. Jokaiseen testausprosessiin kuuluu olennaisena osana testausobjekti eli testauksen kohde. Jotta testaukselle voitaisiin määrittää tietyt kriteerit siitä mitä testataan, täytyy testauksen kohteesta olla selvillä sen odotettu toiminta eli testausperusta, joka määrittelee testausobjektin toiminnallisuuden. Kaikkia testejä ei ole mahdollista eikä myöskään mielekästä suorittaa, joten testaukselle täytyy luoda testausstrategia, jota noudatetaan testauksessa. Testausstrategian määrittelee hyvin pitkälle testauspäällikkö eli johtaja, joka saa ohjeensa ylemmältä tasolta muun muassa käytettävissä olevista resursseista. Testausstrategiaan vaikuttavat myös olennaisesti testausobjektin loppukäyttäjät, jotka määrittelevät testausobjektin lopullisen käyttötarkoituksen. Suorittaakseen testejä testaaja tarvitsee muitakin välineitä kuin testausobjektin käyttöönsä. Testaaja voi tarvita testien suorittamiseen muun muassa testauslaitteiston. Tällaisia testien suorittamiseen tarvittavia erillisiä välineitä kutsutaan testausprosessin infrastruktuuriksi. Kun testaaja suorittaa testausstrategian mukaisia testejä, hän saattaa löytää epäkohtia, kuten esimerkiksi ohjelmointivirheitä tai toiminnallisia virheitä, testausobjektista. Epäkohtien laadusta ja niiden testausobjektin toiminnallisuudelle aiheuttamien vikojen kriittisyydestä päätellen testaaja muodostaa arvion testausobjektin laadusta, mikä antaa kehitysprosessiin osallistuville neuvoja siitä, miten prosessia tulisi jatkaa. [22]



Kuva 15: Testausprosessin yleiset elementit. [22]

Erityisesti tämän työn testausprosessia silmälläpitäen, voidaan kuvan 15 elementit nimetä yksilöllisesti sopimaan koneenohjausyksikön laiteohjelmiston automaattiseen testausprosessiin. Kuvassa 16 on esitetty tämän työn testausprosessin elementit sovitettuna yleisen testausprosessin malliin.



Kuva 16: Koneenohjausyksikön laiteohjelmiston testausprosessin elementit.

4.1.1. Automatisointi

Automatisoimalla testejä voidaan vähentää testaukseen kuluvaa aikaa, mitä kautta voidaan esimerkiksi lisätä suoritettavien testitapausten tai toistojen määrää, testaukseen käytettävissä olevan ajan ollessa vakio. Automaattisen testauksen avulla voidaan myös säästää aikaa ja rahaa sekä parantaa välillisesti testausobjektien laatua, tuottamalla tietoja testausobjektin vaatimustenmukaisuudesta. Lisäksi on olemassa testejä joita ei voi manuaalisesti suorittaa, kuten esimerkiksi erilaiset tilastolliset testit, joiden tulokset kehittyvät ajan kuluessa pitkällä aikavälillä, sekä esimerkiksi kehittyvät algoritmit, joita olisi manuaalisesti käytännössä mahdoton testata, koska niiden toiminta kehittyy kohti optimaalisempaa ratkaisua ajon aikana. Testausautomaation avulla voidaan myös suorittaa automaattisesti erilaisia toistotestejä, joissa samaa testiä toistetaan erilaisilla parametreilla useita kertoja, sekä ajallisesti pitkiä rasitustestejä, joiden avulla voidaan määrittää testausobjektin erilaisia kriittisiä ääriarvoja. [22]

Periaatteessa jokainen suoritettava testi on mahdollista automatisoida. Käytännössä kuitenkin yleensä vain harvat testeistä ovat kokonaan automatisoituja. Testausautomaatio tulee kyseeseen yleensä seuraavanlaisissa tilanteissa: [22]

- Testejä täytyy toistaa usein ja useita kertoja.
- Perustesti täytyy suorittaa useita kertoja erilaisilla parametreilla, koska järjestelmässä testattava toiminto voi saada laajan skaalan erilaisia parametreja.
- Suoritettavat testit ovat hyvin monimutkaisia ja/tai virhealttiita.
- Testaus vaatii monimutkaista laitteistoa, joka tuottaa, vastaanottaa ja analysoi erilaisia signaaleja. Signaalien rinnakkaisuus lisää entisestään tällaisen järjestelmän kompleksisuutta.

Tämän diplomityön automaattisen laiteohjelmiston testausjärjestelmän tarpeen voi perustella kaikilla edellä mainituilla seikoilla.

Testausautomaatiota suunnitellessa täytyy ottaa huomioon monia erilaisia yksityiskohtia. Muutokset järjestelmässä, suunnittelussa ja vaatimuksissa ovat hyvin yleisiä, varsinkin uusia ohjausyksikkömalleja suunniteltaessa. Tällaiset muutokset uhkaavat automaattisen testausjärjestelmän käytettävyyttä ja ne täytyy ottaa huomioon jo järjestelmän suunnittelussa. Muutokset voivat aiheuttaa seuraavia asioita testausjärjestelmälle: [22]

- Uusia testitapauksia
- Muutoksia testitapauksiin
- Erilaisia tuloksia testitapauksista
- Järjestelmän kommunikaatiorajapinnan muutokset
- Toiminnalliset muutokset
- Erilaiset signaalit
- Erilainen sisäinen toteutus

Kuten kaikki tuotekehitysprosessit, myös automaattisen testausjärjestelmän kehitysprosessi voidaan jakaa karkeasti kolmeen eri vaiheeseen. Nämä vaiheet ovat määrittely, toteutus ja hyväksikäyttö. Tämän työn kannalta katsottuna nämä kolme vaihetta käsittävät seuraavat askeleet:

- Määrittely:
  - Laiteohjelmiston automaattisen testausjärjestelmän määrittelyvaiheessa on määriteltävä testausobjekti eli tässä työssä koneenohjausyksikön laiteohjelmisto. Tämän jälkeen täytyy miettiä mitä toimintoja halutaan testata. Tässä työssä testauksen kohteena ovat kaikki ohjausyksikön laiteohjelmiston ohjaamat toiminnot, joita on tarkemmin käsitelty kappaleessa 3. Tämän jälkeen tulee määritellä automaattiselta testauslaitteistolta vaaditut toiminnot, testattavia toimintoja

silmällä pitäen. Määrittelyvaiheessa määritellään myös yksityiskohtaiset testitapaukset testausjärjestelmälle sekä testausstrategia.

- Toteutus:
  - Toteutusvaiheeseen voidaan siirtyä kun laitteiston vaaditut toiminnot on saatu määriteltyä. Laitteistolta vaadittujen toimintojen pohjalta tehdään laitteiston ja testaukseen tarvittavien ohjelmien valinta. Tässä työssä keskeisenä kriteerinä laitteistolle on sen modulaarisuus, joka täytyy ottaa erityisesti huomioon laitteistoa valittaessa. Kun laitteisto ja ohjelmat on valittu, voidaan aloittaa rakennusvaihe. Laitteisto ja siihen kuuluvat lisälaitteet asennetaan toteutusvaiheessa. Tarvittavat johdotukset täytyy myös tehdä, jotta testauslaitteisto voidaan kytkeä testausobjektiin eli ohjausyksikköön. Kun laitteisto ja siihen kuuluvat ohjelmat ovat kunnossa, voidaan aloittaa itse testien implementointivaihe. Testien implementointivaiheessa yksittäiset suoritettavat testit implementoidaan testitapausten määritelmän pohjalta. Toteutusvaihe päättyy, kun koko testausjärjestelmä on implementoitu ja testattu toimivaksi. Testausjärjestelmän toiminnallinen testaus on olennainen osa toteutusvaihetta.

- Hyväksikäyttö:
  - Hyväksikäyttövaiheeseen päästään, kun testausjärjestelmä on valmis käytettäväksi. Tässä vaiheessa järjestelmää käytetään laiteohjelmiston toimintojen automaattiseen testaukseen testausstrategian mukaisesti. Hyväksikäyttövaiheeseen kuluu myös olennaisena osana testausjärjestelmän jatkokehitys ja ylläpito, jotta testausjärjestelmästä saataisiin irti maksimaalinen hyöty mahdollisimman pitkään.

4.2. Ohjelmointi

Automaattisen testausjärjestelmän ohjelmointiin käytetään tässä työssä seuraavissa luvuissa 4.2.1. ja 4.2.2. esitettyjä LabVIEW- ja TestStand-ohjelmistoja. LabVIEW-ohjelmistoa käytetään testauslaitteiston yksittäisten tehtävien ohjelmointiin ja TestStand-ohjelmistoa LabVIEW:llä ohjelmoitujen toimintojen suorittamiseen automaattisesti testaussekvenssinä.

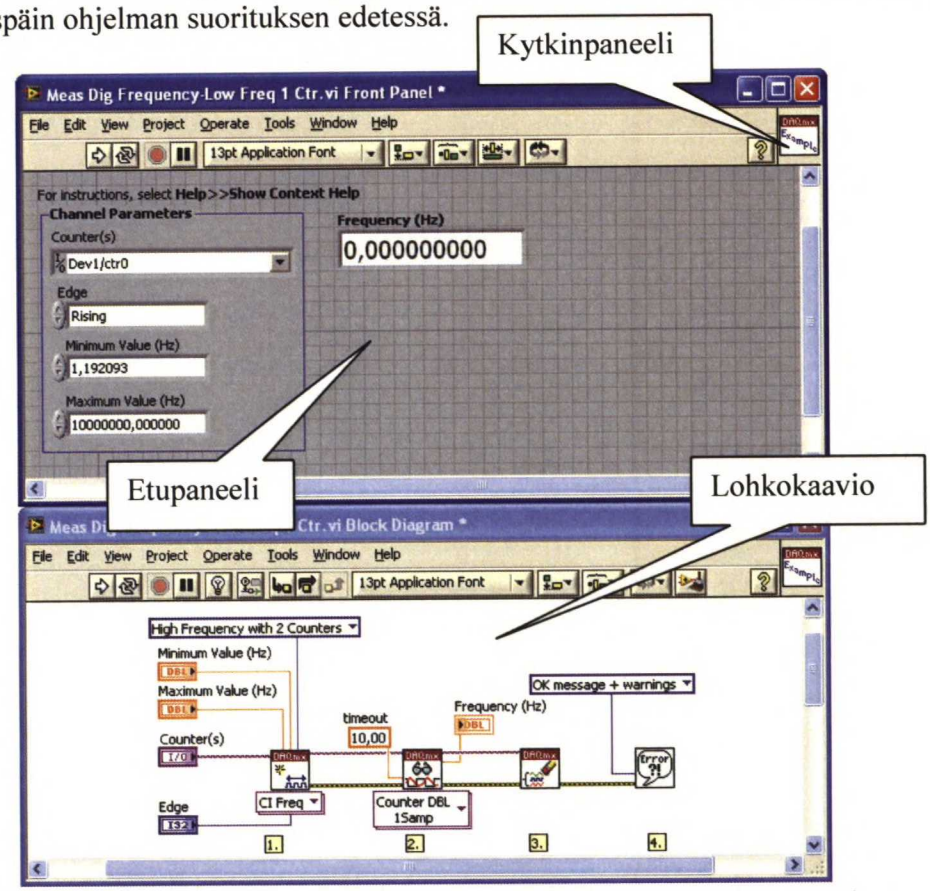
Tässä diplomityössä kehitettävä automaattinen testausjärjestelmä tuottaa ohjelmointiin monia erityispiirteitä. LabVIEW:llä ohjelmoitavat ohjelmamoduulit täytyy ohjelmoida siten, että moduuleista tulee mahdollisimmat yleiskäyttöisiä, mikä parantaa huomattavasti järjestelmän ylläpidettävyyttä ja pidentää järjestelmän elinkaarta. Koska testausjärjestelmällä on tarkoitus testata useita erilaisia ohjausyksiköitä, joiden I/O-rajapinta saattaa vaihdella huomattavasti, on ohjelmamoduulien yleiskäyttöisyys eri ohjausyksiköiden testaussekvenssejä ajatellen todella tärkeää. TestStand:in kannalta on tärkeää, että erilaisten ohjausyksiköiden testaussekvenssejä voidaan luoda samoja LabVIEW:in ohjelmamoduuleja uudelleenkäyttämällä, asettelemalla ainoastaan ohjelmamoduulien parametreja. Tämä

lisää myös osaltaan ohjelmiston ylläpidettävyyttä sekä nopeuttaa järjestelmän ohjelmointia huomattavasti. [22]

4.2.1. LabVIEW

LabVIEW on National Instrumentsin kehittämä ohjelmointiympäristö, joka perustuu graafiseen G-kieleen [23]. LabVIEW-ohjelmia kutsutaan virtuaali-instrumenteiksi, koska niiden ulkoasu ja toiminta jäljittelevät fyysisiä mittausinstrumentteja. LabVIEW-ohjelman rakenne eroaakin melkoisesti tavallisen tekstipohjaisen ohjelman, kuten esimerkiksi C-kielisen ohjelman, rakenteesta. LabVIEW:llä ohjelmoitu ohjelma ei perustu laisinkaan tekstiin, vaan se rakennetaan erilaisista graafisista elementeistä, jotka kuvaavat tiedon virtaamista läpi ohjelman. [24]

Kuvassa 17 on LabVIEWillä ohjelmoitu G-kielinen ohjelma. G-kielinen ohjelma eli virtuaali-instrumentti sisältää kolme komponenttia: lohkokaavion, etupaneelin, ja kytkinpaneelin. Näistä lohkokaaviossa esitetään itse graafinen ohjelmakoodi, etupaneelia käytetään käyttäjärajapintana itse ohjelmaan ja kytkinpaneelin avulla ohjelmakomponentti voidaan esittää toisessa ohjelmassa [25]. Tieto virtaa kuvassa vasemmalta oikealle ja ylhäältä alaspäin ohjelman suorituksen edetessä.



Kuva 17: G-kielinen LabVIEW-ohjelma eli virtuaali-instrumentti.

LabVIEW on muodostunut lähes standardin kaltaiseksi, kun puhutaan mittaus- ja testaussovellusten ohjelmoinnista. Helppokäyttöisyytensä, nopeutensa ja tehokkuutensa ansiosta G-kieli soveltuu tietyin varauksin myös yleisohjelmointikieleksi. Jotkin operaatiot, kuten esimerkiksi merkkijonojen käsittely, ovat LabVIEW:llä vaikeita ja monimutkaisia toteuttaa. G-kielen soveltuvuutta yleisohjelmointikieleksi haittaa myös se, että LabVIEW:llä ohjelmoidut ohjelmat vaativat toimiakseen LabVIEW run-time enginen, jota ei ole asennettuna valmiiksi yhteenkään yleiseen käyttöjärjestelmään, toisin kuin esimerkiksi C-kielisen ohjelman vaatimat kirjastot, jotka löytyvät useista yleisistä käyttöjärjestelmistä valmiiksi asennettuina. Erilaisten testaus- ja mittaussovellusten ohjelmointia varten LabVIEW tarjoaa todella paljon monipuolisia valmiita komponentteja, joiden avulla esimerkiksi monimutkaisten testaussovellusten ohjelmointi helpottuu ja nopeutuu huomattavasti verrattaessa tekstipohjaisiin, kuten C-kieleen perustuviin, ohjelmointiympäristöihin. [23, 26]

4.2.2. TestStand

National Instrumentsin kehittämä TestStand on testien hallintaan kehitetty ohjelmisto. TestStand on kehitetty erityisesti automaattisen testauksen vaatimuksia ja tarpeita silmälläpitäen. Sillä onnistuu niin testiohjelmiston kehitys kuin myös ajaminen nopeasti ja vaivattomasti. TestStand mahdollistaa eri ohjelmointikielillä tehtyjen ohjelmamoduulien ajamisen samassa testisovelluksessa osana samaa testisekvenssiä, mikä mahdollistaa myös muiden kuin LabVIEW-ohjelmiston hyödyntämisen tarpeen niin vaatiessa. Tällainen tarve saattaa tulla eteen esimerkiksi tilanteissa, joissa LabVIEW:in käyttö jonkin tietyn operaation tekemiseen ei ole järkevää, esimerkiksi merkkijonojen käsittely, tai kun halutaan uudelleen käyttää ohjelmistokomponentteja, jotka on alun perin kirjoitettu jollain toisella ohjelmointikielellä ja mahdollisesti johonkin eri tarkoitukseen. Automaattisen testauksen tarpeiden mukaisesti TestStand tarjoaa myös mahdollisuuden määritellä testisekvenssiin erilaisia ehtoja ohjelmamoduulien suoritukselle, raportointiominaisuudet monissa eri tiedostomuodoissa, erilaisten muuttujien ja tiedon tallentamisen tietokantaan sekä helposti määriteltävän ja muokattavan käyttäjärajapinnan. [27]

4.3. Testauslaitteisto

Automaattinen testaus tuottaa myös testausjärjestelmän laitteistolle tiettyjä erityisvaatimuksia. Jotta automaattisesti suoritettava testaussekvenssi olisi mahdollinen, täytyy testauslaitteiston jokainen komponentti olla ohjelmallisesti ohjattavissa, mikä tarkoittaa tämän työn kannalta LabVIEW-ajureiden olemassaolon vaatimusta. Laitteiston ytimen muodostaa kontrolleri eli käytännössä tietokone, joka suorittaa testaussekvenssiä TestStand:in avulla. TestStand:in testaussekvenssin yksittäiset ohjelmamoduulit ohjelmoidaan tässä työssä LabVIEW:llä. LabVIEW:llä ohjelmoidut ohjelmamoduulit ohjaavat kaikkia testauslaitteiston osia, kuten I/O-kortteja ja jännitelähteitä, testaussekvenssin määrittelyn mukaisesti.

4.3.1. Automaattinen laiteohjelmiston testauslaitteisto

Tämän työn testauslaitteiston rungon muodostaa seuraavassa luvussa 4.3.2. esitetty PXI-kehikko. PXI-kehikkoon on saatavilla paljon erilaisia kommunikaatioon ja I/O-signaalien generointiin tarkoitettuja valmiita kortteja, minkä takia se on valittu testauslaitteiston rungoksi. PXI tarjoaa myös mahdollisuuden modulaariseen laitteistosuunnitteluun ja rakentamiseen, koska PXI-kehikkoon on helppo kytkeä uusia kortteja ja lisälaitteita myöhemmin tarvittaessa. Modulaarinen laitteisto on yksi tämän työn tärkeimmistä vaatimuksista. Kuten aiemmin on mainittu, täytyy kaikki laitteiston yksittäiset komponentit olla ohjelmallisesti ohjattavissa, jotta automaattinen testaus olisi mahdollista. Tämän takia kaikki laitteiston komponentit on valittu siten, että niiden ohjaus onnistuu ohjelmallisesti PXI-kehikossa sijaitsevan kontrollerin ja siihen asennettujen ohjelmistojen avulla.

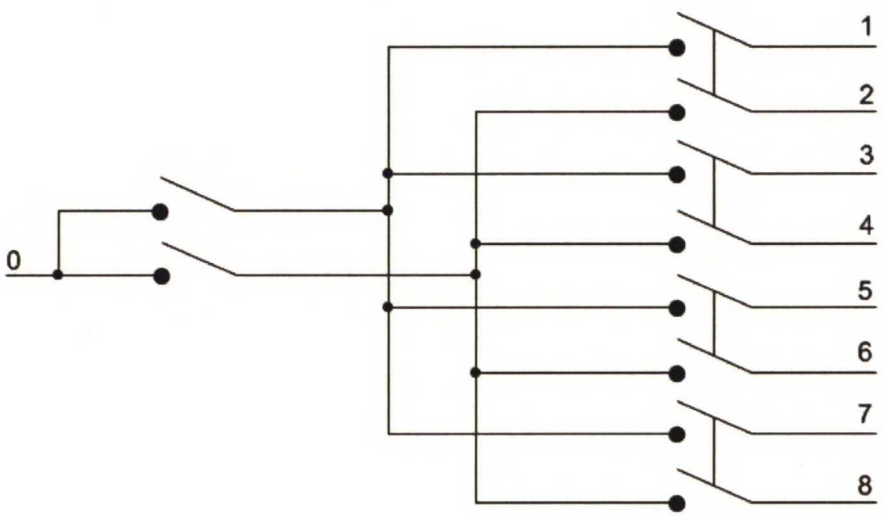
Luvussa 3. on esitetty laiteohjelmiston ohjaamat toiminnot ohjausyksikössä. Jotta näitä toimintoja voitaisiin automaattisesti testata, täytyy testauslaitteiston tarjota mahdollisuudet kaikkien ohjausyksiköiden I/O-signaalien luontiin ja vastaanottamiseen. Seuraavassa on esitetty vaatimukset testauslaitteiston I/O-rajapinnalle:

- Digitaalisia tuloja
- Digitaalisia lähtöjä
- Analogisia virtalähtöjä
- Analogisia jännitelähtöjä
- PWM-liityntä taajuuden ja pulssinleveyden mittaamiseen
- CAN-liityntä
- RS-232-liityntä
- RS-485-liityntä
- Jännitelähtö ohjausyksikön käyttöjännitettä varten

Näistä signaaleista kaikki muut paitsi käyttöjännite ohjausyksikölle tuotetaan PXI-kehikkoon liitettävillä korteilla. Ohjausyksikön käyttöjännitteen tuottamista varten testauslaitteistoon tarvitaan ulkoinen ohjelmoitava teholähde, koska PXI-kehikkoon liitettävien jännitekorttien tuottama maksimilähtöteho on pieni ja ei välttämättä riitä tuottamaan ohjausyksikölle riittävästi tehoa.

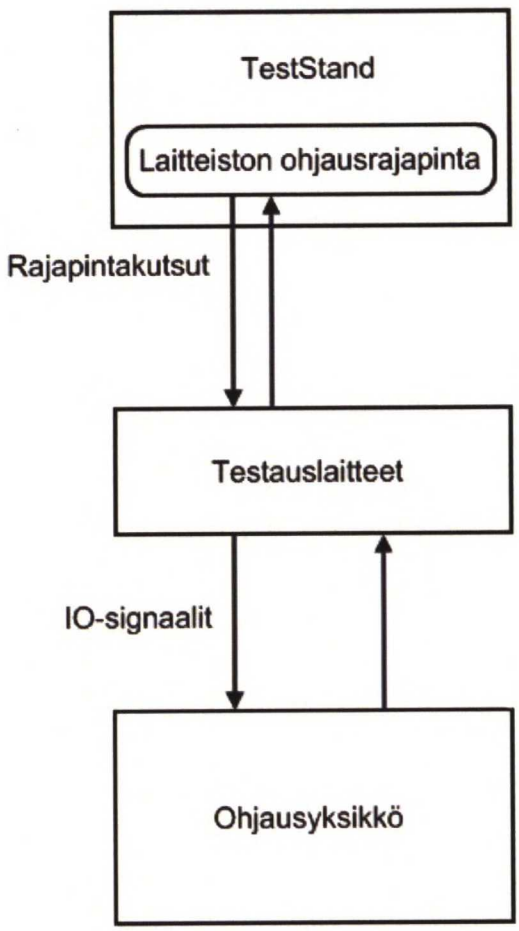
Testauslaitteiston täytyy soveltua myös erilaisten ohjausyksiköiden testaukseen, joissa on erilaisia pinnimääritelmiä eli käytännössä laitteiston on kyettävä tuottamaan ohjausyksikön pinneihin erilaisia signaaleja siten, että manuaalisia kytkentämuutoksia ei tarvitse tehdä. Tämän erityispiirteen takia täytyy testauslaitteistossa olla myös multiplekseri-kortteja, joita voidaan ohjelmallisesti ohjata. Kuvassa 18 on esitetty tässä työssä käytettävien multiplekseri-korttien toimintaperiaate. Testauslaitteistossa käytetään 8-1-multipleksereitä, joiden avulla voidaan kytkeä 8 erilaista signaalia yhteen pinniin. Käytännössä tämä toimii siten, että kuvan 18 kanaviin 1-8 voidaan kytkeä signaalit: digitaalitulo, digitaalilähtö, analogiajännitelähtö, analogiavirtalähtö, pulssilähtö ja PWM-mittaus. Näistä signaaleista

voidaan ohjelmallisesti valita haluttu suure ohjausyksikön pinniin, joka on kytketty multiplekserin kanavaan 0. Muut kytkennät, kuten CAN-väyläkytkentä, kytketään ohjausyksikköön suoraan ilman multipleksereitä, koska kaikissa testauksen kohteena olevissa moduuleissa on samanlaiset kytkennät näitä varten. [28]



Kuva 18: 8-1-multiplekserin toimintaperiaate. [28]

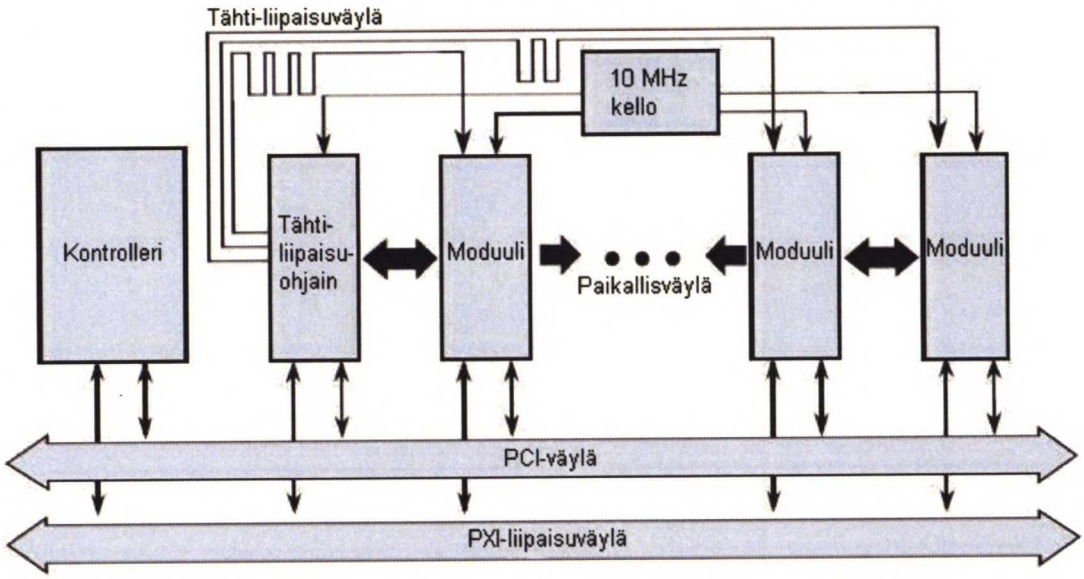
Kuvassa 19 on esitetty kuinka TestStand käyttää testauslaitteistoa suorittaessaan testaussekvenssiä. TestStand:in testaussekvenssi koostuu ohjelmamoduuleista, jotka on ohjelmoitu LabVIEW:llä. LabVIEW-ohjelmat ohjaavat laiteajureiden avulla laitteiston ohjausrajapinnan kautta testauslaitteiston kaikkia toimintoja testiohjelmoijan haluamalla tavalla. TestStand suorittaa näitä ohjelmia ohjelmoidun testisekvenssin mukaisessa järjestyksessä automaattisesti. Kun kaikki yksittäiset testaustoiminnot on ohjelmoitu, niin koko testausprosessi on periaatteessa mahdollista suorittaa automaattisesti. I/O-signaalien kytkentä testauslaitteiston korteista testattavaan ohjausyksikköön tapahtuu edellä mainitulla tavalla, joko suoraan tai multiplekserien kautta.



Kuva 19: Testausjärjestelmän rajapinnat.

4.3.2. PXI – "PCI extensions for instrumentation"

PXI on CompactPCI:in eli teollisuus- ja tutkimuslaitoskäyttöön koteloidun PC-tekniikan versio, joka on kehitetty erityisesti mittaus-, ohjaus- ja testaustekniikkaa silmällä pitäen. Modulaarinen PXI-tekniikka on ollut olemassa jo vuodesta 1997. Mekaanisesti se perustuu CompactPCI-arkkitehtuuriin, mutta se tarjoaa monipuolisempia toimintoja muun muassa synkronointien ja liipaisujen saralla. Kuvassa 20 on kuvattu PXI-arkkitehtuurin synkronointi- ja liipaisuväylät. PXI tarjoaa sisäänrakennetun 10 MHz:n kellon, jota kaikki PXI-laitteiston moduulit voivat käyttää yhteisenä kellona keskinäisten toimintojensa, kuten samanaikaisten mittausten, synkronoinnissa. Lisäksi arkkitehtuuri tarjoaa tähteen kytketyn liipaisuväylän sekä PXI-liipaisuväylän, joita laitteiston moduulit, esimerkiksi mittauskortit, voivat käyttää yhteisten toimintojensa synkronointiin liipaisusignaalien kautta. [29]



Kuva 20: PXI-arkkitehtuurin synkronointi- ja liipaisuväylät. [30]

PXI on avoin teollisuusstandardi, jota hallinnoi PXISA-järjestö. PXI tarkoittaa käytännössä teollisuus-PC:tä, johon voidaan vaivattomasti kytkeä erilaisia signaali- ja mittauskortteja nopean väylän kautta. PXI:n tapauksessa nopea väylä tarkoittaa PCI-väylää, mutta nykyään kehitystyön tuloksena PXI:stä on myös tarjolla uudempi versio PXIe, jonka väylänä toimii uudempi PCI Express. PCI-väylä (Peripheral Component Interconnect) on alun perin Intelin kehittämä, myöhemmin PCI-standardin mukainen rinnakkaismuotoinen tietokone-väylä, jonka avulla liitetään lisälaitteita tietokoneeseen. [31]

Normaaliin PC-ympäristöön verrattuna PXI tarjoaa monia eri hyötyjä teollisuutta ja tutkimuslaitoskäyttöä silmälläpitäen. Näitä hyötyjä ovat muun muassa:

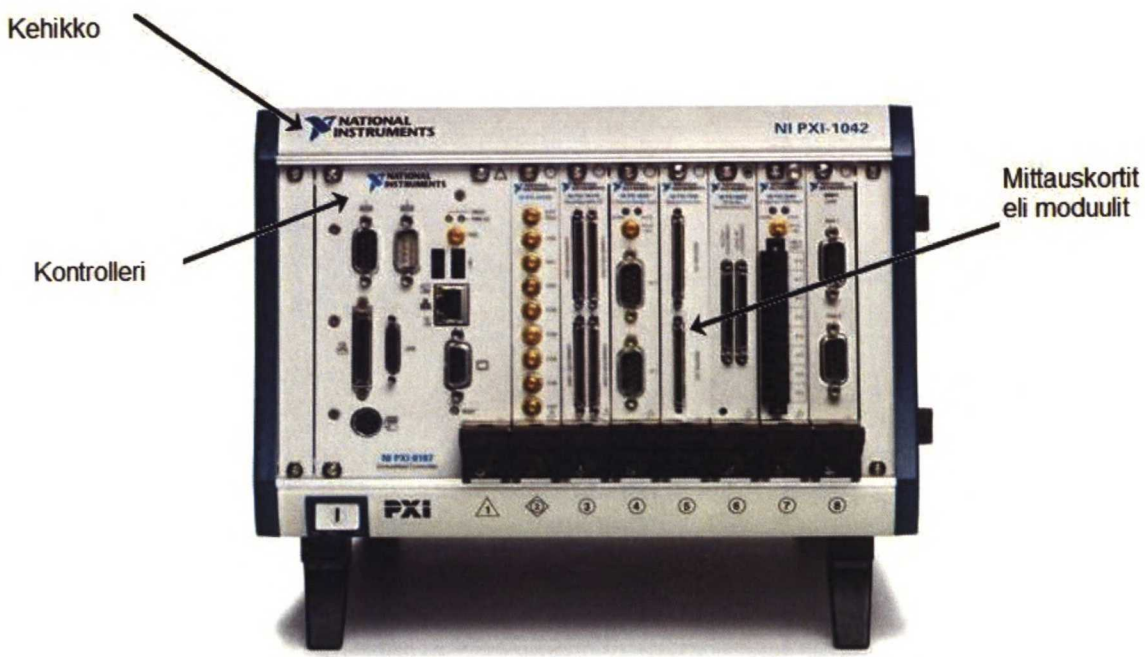
- Pieni ja tiivis teollisuuskäyttöön kehitetty paketti
  - Laitteisto on suhteellisen pieni korttien I/O-määrään nähden ja kontrollerin osuus laitteiston koosta on hyvin pieni. Pienuus toisaalta myös rajoittaa korttien ja I/O:n määrää yksittäisessä laitteistossa. Toisaalta laitteistoja voidaan helposti myös ketjuttaa.

- Avoin standardi
  - PXI-laitteisto pohjautuu avoimeen PCI-väylästandardiin ja myös PXI-alusta on avoin, mistä johtuen monet laitevalmistajat ovat tuoneet korttejaan ja laitteistojaan markkinoille, mikä tuottaa lisää vaihtoehtoja valintoihin.

- Laaja valikoima erilaisia instrumentteja ja moduuleja
  - Laaja valikoima johtuu lähinnä siitä, että PXI on avoin alusta, jota kaikki halukkaat laitevalmistajat voivat hyödyntää.

- Laajennettavuus
  - PXI-laitteisto on helposti laajennettavissa hankkimalla uusia kortteja ja liittämällä ne kehikkoon. PXI-kehikkojen ketjutus on helppoa, kun yhdestä kehikosta loppuu tila. Huonona puolena PXI-kehikon koko rajoittaa korttien kokoa ja tätä kautta yhden kortin mahdollista I/O-määrää, mitä kautta korttien määrä muodostuu helposti suureksi jo ennen laajennustarvetta.

Kuvassa 21 on esitetty PXI-kehikko, jossa on kytkettynä kontrolleri ja erilaisia mittauskortteja. PXI-spesifikaation mukaisesti kontrolleri sijaitsee vasemmalla PXI-kehikossa. Kontrollerissa sijaitsee muun muassa massamuisti sekä käyttöjärjestelmä, joka voi olla esimerkiksi Windows 2000/XP tai vaihtoehtoisesti reaaliaikakäyttöjärjestelmä kuten LabVIEW Real-Time. Erilaiset liitynnät etäohjaukselle sekä esimerkiksi näytölle sijaitsevat kontrollerin etupaneelissa. Liityntämahdollisuuksia on myös mahdollista lisätä käyttämällä erilaisia moduuleja. Kontrollerista oikealle tulevat erilaiset käyttäjän valitsemat moduulit liityntöineen. [30]



Kuva 21: 8-paikkainen PXI-kehikko, jossa kytkettynä kontrolleri (PC) ja erilaisia mittauskortteja. [30]

Tämän työn automaattisessa testausjärjestelmässä käytetään PXI-laitteistoa, johon on liitetty erilaisia I/O-kortteja ohjausyksiköiden I/O-liityntöjen vaatimusten mukaisesti. PXI-laitteistossa on myös kontrolleri, jossa käytetään Windows XP-käyttöjärjestelmää. Windows XP-käyttöjärjestelmään on asennettu ohjelmamoduulien ohjelmointiin ja testi-sekvenssin luontiin käytettävät ohjelmistot LabVIEW ja TestStand.

5. Työn toteutus

Tässä luvussa esitellään testausjärjestelmän toteutus. Ensimmäiseksi käydään läpi testausjärjestelmän kehitysvaiheet sekä laitteiston toteutus laitevalinnoista ohjelmointiin. Tämän jälkeen esitellään testausjärjestelmän käyttöä ohjausyksiköiden testauksessa. Luvun lopuksi pohditaan vielä testausjärjestelmän ylläpitoon ja elinkaareen liittyviä tekijöitä tämän työn toteutuksen kannalta.

5.1. Testausjärjestelmän toteutus

Testausjärjestelmän kehittäminen ja suunnittelu aloitettiin määrittelemällä aluksi testattavat toiminnot, joita automaattisella testausjärjestelmällä halutaan testata. Testattavien toimintojen määrittelyn avulla saatiin tarpeelliset tiedot järjestelmään vaadittavista komponenteista ja järjestelmän toiminnoista eli järjestelmälle asetetut vaatimukset. Seuraavassa on esitetty laiteohjelmiston toimintaan liittyvät testausjärjestelmällä testattavat perustoiminnot:

- Flash-muistin ohjaus
  - Tallennus
  - Tyhjennys
  - Parametrien lukeminen ja kirjoittaminen
  - Kommunikointiparametrien toiminta
- Ohjausyksikön parametrien toiminta
- I/O-liityntöjen toiminta
  - Pulssinleveysmoduloidut lähdöt
  - Digitaaliset tulot
  - Digitaaliset lähdöt
  - Analogiset tulot
  - Pulssilaskuritulot
  - Taajuuslaskuritulot
- CAN
  - Kommunikaatio
  - Kommunikointinopeudet
  - Väylien kuormitus
- CANopen
  - Objektikirjaston toiminta
  - SDO-protokollan toiminta
  - PDO-protokollan toiminta
  - NMT-protokollan toiminta
- RS-232-liitynnän ja -kommunikaation toiminta
- RS-485-liitynnän ja -kommunikaation toiminta

Laiteohjelmiston perustoimintojen tarkemmat määritelmät löytyvät luvuista 3. ja 4. Näiden laiteohjelmiston testaukselle asetettujen vaatimusten avulla voitiin määritellä kaikki järjestelmältä vaadittavat toiminnallisuudet sekä testauslaitteiston I/O-rajapinta. Luvussa 5.1.1. on esitetty testausjärjestelmän laitteiston komponentit yksityiskohtaisesti. Automaattiseen testaukseen on tarjolla runsaasti erilaisia valmiita laitteistoratkaisuja. Tällaisten laitteistojen hyvä puoli on, että ne on valmiiksi rakennettu toimintavalmiiksi ja automaattisen testauksen näkökulmasta toteutettavaksi jää ainoastaan laitteiston ohjelmointi.

Testausjärjestelmän komponenttien määrittelyvaiheessa painotettiin tässä työssä erityisesti laitteiston modulaarisuutta ja helppoa laajennettavuutta, minkä takia työn toteutuksessa ei päädytty käyttämään valmiita laitteistoratkaisuja. Valmiiden laitteistokokonaisuuksien heikkous on yleensä niiden huono laajennettavuus, johtuen komponenttien integraatiosta. Lisäksi valmiiksi toteutetuissa laitteistoissa on yleensä ennalta määrätty I/O-rajapinta, jossa saattaa olla useitakin erilaisia I/O-toimintoja, joita ei tässä työssä tarvittu. Ylimääräiset I/O-toiminnot aiheuttaisivat järjestelmälle turhia kustannuksia.

Laitteiston komponenttien päätoimittajaksi valittiin National Instruments johtuen pitkälti yrityksen vahvasta asemasta testauslaitteistojen alalla. National Instruments on LabVIEW-kehitysympäristön kehittäjä, joka vaikutti myös osiltaan laitevalintoihin. LabVIEW ja TestStand ovat molemmat National Instrumentsin kehittämiä ohjelmistoja, mistä johtuen niiden integraatio on vaivatonta ja nopeaa, minkä ansiosta testisekvenssin ajoon TestStand oli luontainen valinta, kun testausohjelmien kehitys oli päätetty tehtäväksi LabVIEW:llä. LabVIEW:n valinta ohjelmakomponenttien toteutukseen oli myös luontainen, koska LabVIEW on tarkoitettu juuri tässä työssä kehitettävän testausjärjestelmän kaltaisten järjestelmien ohjaukseen ja vastaavia kokonaisvaltaisia, helposti omaksuttavia sekä nopeita, sovelluskehitysympäristöjä ei markkinoilta ole saatavissa.

PXI-arkkitehtuuri tarjoaa testausjärjestelmälle todella hyvän laajennettavuuden, koska PXI-kehikkoon voidaan lisätä helposti erilaisia I/O-kortteja yksi kerrallaan tarpeen vaatiessa. Lisäksi PXI-kehikkoja on mahdollista ketjuttaa, jos yhdestä kehikosta loppuvat korttipaikat. PXI-arkkitehtuuri tarjoaa täten hyvän ratkaisun laitteiston modulaarisuus- ja laajennettavuusvaatimukseen.

Kaikkia määriteltyjä komponentteja ei kuitenkaan National Instrumentsin valikoimasta löytynyt, joten osa järjestelmän komponenteista hankittiin muilta valmistajilta määriteltyjen ominaisuuksien perusteella. Ainoat luvun 4.3.1. kuvaaman toimintaperiaatteen mukaisesti toimivat multiplekserit löytyivät Pickeringin valikoimista. Multiplekserille oli välttämätöntä, että ne voidaan kytkeä PXI-kehikkoon ja että niitä voidaan ohjata LabVIEW:n avulla. Lisäksi multiplekserien 8-1 kytkentäperiaate määriteltiin vaatimukseksi jo ennen multiplekserien toimittajan valintaa.

Ohjelmoitavien jännitelähteiden kytkentä testausjärjestelmän osaksi täytyi sujua vaivattomasti ja lähteiden ohjaus oli onnistuttava LabVIEW:n avulla. Jännitelähteille määritellyt raja-arvot olivat alkumääritelmien mukaan:

- Lähde 1: Jännite yli 30 V ja virta yli 10 A.
- Lähde 2: Jännite yli 30 V ja virta yli 5 A.

Näiden määritelmien mukaiset ohjelmoitavat jännitelähteet päädyttiin hankkimaan Instekiltä. Instekillä oli valikoimissaan tarjolla sopivat lähteet, joihin oli olemassa LabVIEW-ajurit. Lähteet olivat myös vaihtoehdoista halvimpia ja niille oli tarjolla edustus Suomessa, mikä on tärkeää huoltovarmuuden kannalta. Lähdettä 1 käytetään testausjärjestelmässä ohjausyksikön käyttöjännitteen (normaalitilassa +24 V) ja PXI-kehikon korttien käyttöjännitteiden tuottamiseen. Lähdettä 2 käytetään jännitteen tuottamiseen ohjausyksikön pinneihin.

Järjestelmän komponenttien hankinnan jälkeen siirryttiin testausjärjestelmän rakennusvaiheeseen. Rakennusvaiheessa toteutettiin testausjärjestelmän laitteiston kokoonpano. PXI-kehikossa oli valmiiksi asennettuna National Instrumentsilta tilatut kortit. Kehikkoon täytyi asentaa vielä Pickeringin multiplekserit jälkikäteen, mikä onnistui PXI-kehikkoon helposti, työntämällä ainoastaan kortit korttipaikkoihin ja lukitsemalla ne ruuvilla paikoilleen. PXI-kehikon asennuksen jälkeen kehikossa sijaitsevaan kontrolleriin liitettiin kaikki oheislaitteet, joita olivat näyttö, näppäimistö, hiiri, ohjelmoitavat jännitelähteet ja ulkoinen CD-asema. Oheislaitteiden asennuksen jälkeen rakennusvaiheessa oli enää tehtävänä johdotukset PXI-kehikon korteista multipleksereihin ja tämän jälkeen multipleksereistä ohjausyksikön pinneihin. Luvussa 5.1.2. on esitetty tarkempi kuvaus testausjärjestelmän liitynnöistä. Testausjärjestelmän liityntöjen toteutuksen jälkeen, kaikki liitynnät testattiin asettelemalla lähtökorteilta määrättyjä arvoja ja syöttämällä tulokorteille määrättyjä arvoja ja tämän jälkeen mittaamalla arvojen oikeellisuudet oskilloskoopilla. Tässä vaiheessa kaikki mahdolliset signaalien kytkentäkombinaatiot käytiin läpi, jotta voitiin varmistua siitä että järjestelmän jokainen liityntä toimii halutulla tavalla. Järjestelmän kaikki testausvaiheet on kuvattu tarkemmin luvussa 5.1.4.

Testausjärjestelmän testausvaiheen jälkeen siirryttiin suunnittelemaan testaussekvenssiä, joka toteutettiin TestStand-ohjelmistolla. Käytännössä tämä tarkoittaa, että suunniteltiin määritelty järjestys, missä erilaiset testit ajetaan laiteohjelmiston toimintojen testaamiseksi. Nämä toiminnot on määritelty tarkemmin tämän kappaleen alussa. Luvussa 5.1.3. on kuvattu tarkemmin testisekvenssi ohjausyksiköiden laiteohjelmiston testaamiseksi. Testisekvenssin määrittely sisälsi myös sekvenssin osien eli LabVIEW-kehitysympäristöllä toteutettavien ohjelmamoduulien toiminnallisen kuvauksen, jonka pohjalta yksittäiset ohjelmamoduulit toteutettiin ja testattiin.

Testisekvenssin ja ohjelmamoduulien määrittelyn valmistuttua oli luontevaa siirtyä ohjelmointivaiheeseen, jossa ohjelmoitiin LabVIEW:llä erilliset ohjelmamoduulit sekä toteutet-

tiin testisekvenssi TestStand:llä. Luvussa 5.1.3. on kuvattu tarkemmin testausjärjestelmän ohjelmointia.

5.1.1. Laitteisto

Testausjärjestelmään valittiin seuraavat, taulukossa 3 esitetyt, komponentit. Taulukon komponentit on eritelty valmistajan mukaan, järjestyksessä National Instruments, Pickering ja viimeiseksi Instek. Kuten edellä jo mainittiin, suurin osa järjestelmän komponenteista valittiin National Instrumentsin valikoimista. Paras vaihtoehto olisi ollut, jos kaikki järjestelmän komponentit olisi voitu tilata suoraan National Instrumentsin valikoimista, jolloin komponenttien tuki ja yhteensopivuus olisi ollut paras mahdollinen. Kaikkia järjestelmän vaatimuksia ei saatu kuitenkaan täytettyä National Instrumentsin tuotteiden avulla, joten hankinnoista osa oli tehtävä muualta. Järjestelmään hankittiin multiplekserit ja niihin kuuluvat kytkentätarvikkeet Pickeringiltä sekä ohjelmoitavat jännitelähteet ja niihin kuuluvat tarvikkeet Instekiltä.

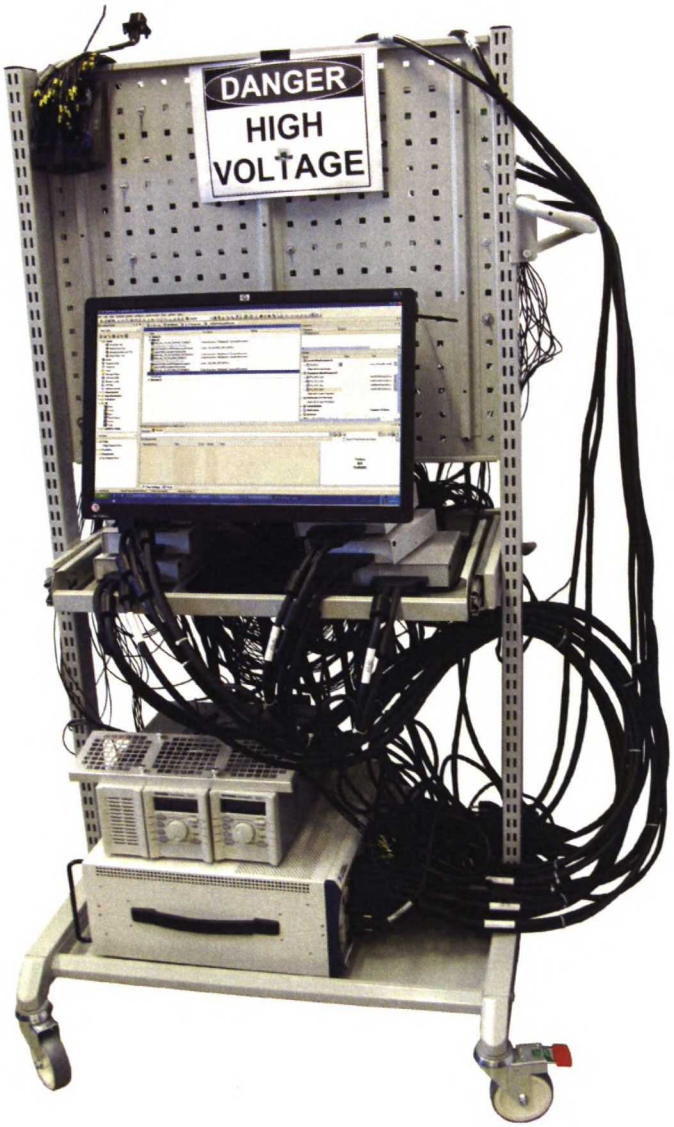
Taulukko 3: Testausjärjestelmän komponentit.

National Instruments		
Malli	**Kuvaus**	**Määrä**
Kontrolleri:		
NI PXI-8106	NI PXI-8106 Core 2 Duo 2.16 GHz Controller, Windows XP	1
1 GB	1 GB DDR2 RAM for NI 8106 and NI 8105 Controller	2
Kortit:		
NI PXI-6511	NI PXI-6511 Industrial 64 DI, Bank Isolated Digital Input	1
NI PXI-6512	NI PXI-6512 Industrial 64 Source DO, Bank Isolated DO	1
NI PXI-6513	NI PXI-6513 Industrial 64 Sink DO, Bank Isolated DO	1
NI PXI-6514	NI PXI-6514 Industrial 32 DI, 32 Source DO Isolated DIO	1
NI PXI-6515	NI PXI-6515 Industrial 32 DI, 32 Sink DO Bank Isolated DIO	1
NI PXI-6624	NI PXI-6624 Industrial 8 Ch-Ch Isolated 48V Counter/Timer	1
NI PXI-6704	NI PXI-6704 DC Analog Output Module	1
NI PXI-8464/2	NI PXI-8464 Software Selectable Series 2 CAN, 9 Pin D-Sub	1
NI PXI-8231	NI PXI-8231, GBIT Ethernet Controller, for Windows	1
NI PXI-8433/2	PXI-8433/2, 2000V Isolated RS422/485, 2 Port Serial Interface	1
Kytkentäkaapelit:		
SH68-68-D1 Cable	SH68-68-D1 Shielded Cable, 2 m	1
SH100-100-F Cable	CABLE ASSY, Type SH100-100-FLEX, 2 m	6
Ruuviterminaalit:		
SCB-100	SCB-100 Noise Rejecting, Shielded I/O Connector Block	6
SCB-68	SCB-68 Noise Rejecting, Shielded I/O Connector Block	1
Kehikko:		
Universal 240VAC	Power Cord, 240V, 10A, Euro, Right Angle	1
NI PXI-1045	NI PXI-1045, 18-Slot 3U Chassis with Universal AC Power Supply	1
Ohjelmistot:		
	NI Developers Suite with Automated Test Option	1

Pickering		
Malli	**Kuvaus**	**Määrä**
Multiplekserit:		
40-615	Very high density multi-banked multiplexer, 20Banks;8Channel;1Pole	4
40-971-200-2M	Cable / Kytkentäkaapeli	4
40-967-550-M	Connector Block / Ruuviterminaali	16

Instek		
Malli	**Kuvaus**	**Määrä**
Ohjelmoitavat jännitelähteet:		
PSH-6012	Programmable Switching DC Power Supply 60V / 12 A / GPIB Interface	1
PSH-6006	Programmable Switching DC Power Supply 60V / 6 A / GPIB Interface	1

Kuvassa 22 on esitetty koko testausjärjestelmä rakennettuna helposti liikuteltavaan kärryyn. Järjestelmä on rakennettu kärryyn, jotta se voidaan siirtää vaivattomasti paikasta toiseen järjestelmän käyttäjän mukana. Kuvassa näkyy alhaalla PXI-kehikko, jonka päällä sijaitsevat molemmat ohjelmoitavat jännitelähteet. PXI-kehikosta lähtevät kaapelit menevät näytön kanssa samalla tasolla oleviin National Instrumentsin ruuviterminaaleihin, jotka on lähemmin esitetty kuvassa 24. Näistä ruuviterminaaleista lähtee johdotukset Pickeringin ruuviterminaaleihin, jotka on esitetty tarkemmin kuvassa 25. Kuvassa 22 Pickeringin ruuviterminaalit sijaitsevat kärryn takapuolella. Pickeringin ruuviterminaaleista kaapelit tulevat takaisin PXI-kehikkoon, jossa ne on liitetty multipleksereihin. Kuvan vasemmassa yläkulmassa näkyy ohjausyksikkö kiinnitettynä järjestelmään, kuvan 26 esittämällä tavalla, testausta varten.



Kuva 22: Testausjärjestelmän laitteisto.

5.1.2. Liitynnät

Automaattisen testausjärjestelmän liityntöjen määrittelyssä käytettiin referenssinä testaukselle kuvassa 3 esitettyä Epec 2024-ohjausyksikköä. 2024-ohjausyksikkö sisältää 69 I/O-pinniä, joihin tuli olla mahdollista tuottaa mitä tahansa testausjärjestelmän I/O-suuretta automaattisesti, ilman manuaalisia erityiskytkentöjä. Lisäksi 2024-ohjausyksikkö sisältää 8 I/O-pinniä, joissa on kiinteisiin paikkoihin määritellyt kaksi CAN-liityntää sekä ohjausyksikön käyttöjännite- ja maaliitynnät.

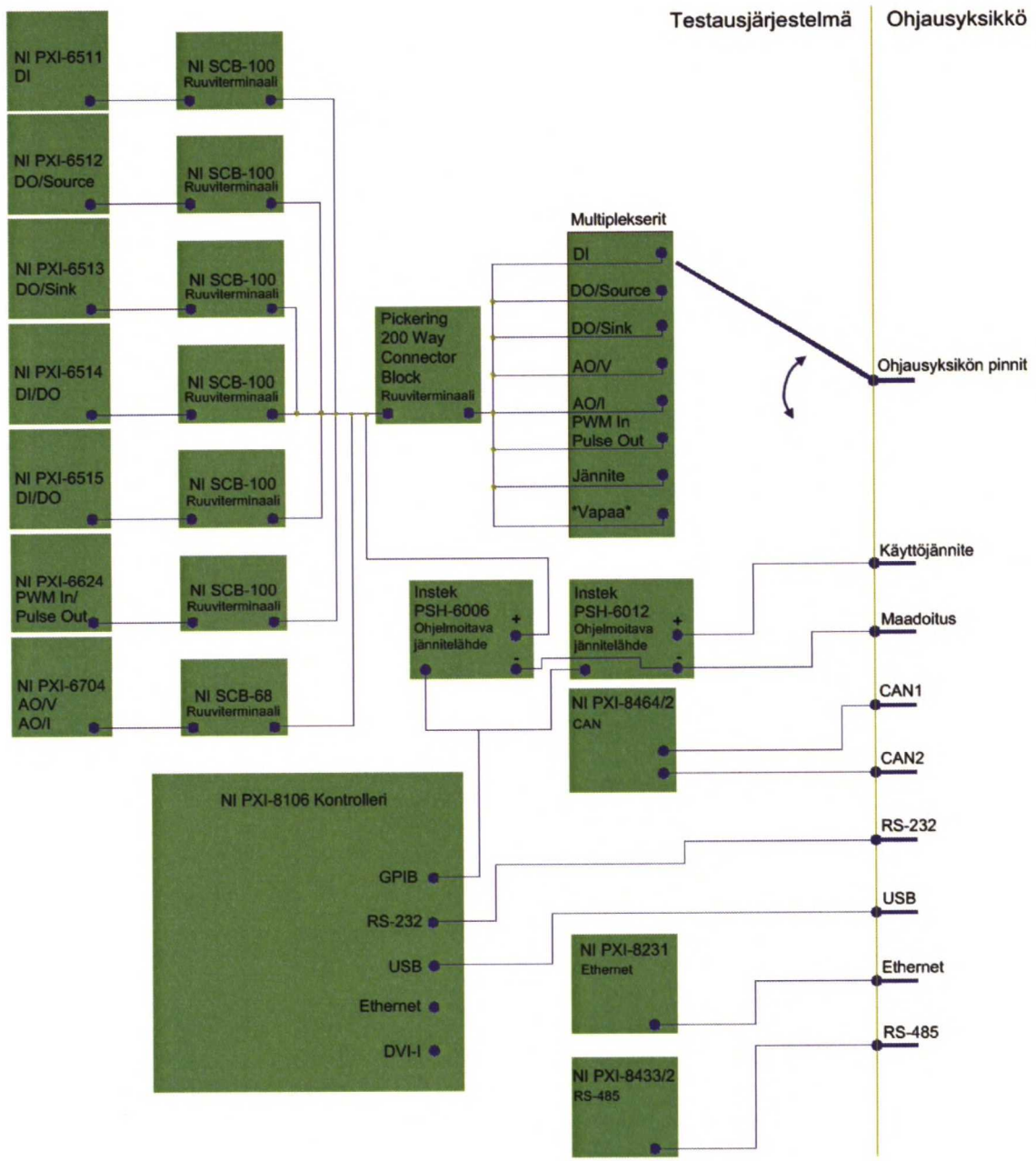
69 I/O-pinniä pidettiin suurimpana mahdollisena I/O-määränä testausjärjestelmälle sitä suunniteltaessa. Järjestelmällä testataan myös ohjausyksiköitä, joissa ei ole 69 I/O-pinniä, vaan niitä saattaa olla vähemmän. Se, että I/O-liityntöjä on järjestelmässä liikaa joillekin ohjausyksikkötyypeille, ei ole kuitenkaan ongelma, koska tällöin voidaan ohjelmallisesti määrittää joitakin testausjärjestelmän pinnejä pois käytöstä.

Kuvassa 23 on esitetty testausjärjestelmään tehdyt fyysiset kytkennät eri laitteiston osien välillä. Taulukosta 3 löytyy piirroksen eri komponenttien tarkemmat kuvaukset mallinumeroiden ja nimien lisäksi. Kuvassa on esitetty PXI-kehikkoon liitetty kontrolleri NI PXI-8106, joka ohjaa kehikkoon liitettyjä komponentteja LabVIEW-ohjelmoinnin mukaisesti. Kontrollerista löytyy joitakin perusliityntöjä valmiiksi, jotka on myös merkitty kuvaan. Kontrollerin lisäksi PXI-kehikosta löytyy kaikki I/O-moduulit, joita testausjärjestelmään on määritelty. Kortit NI PXI-6511, NI PXI-6512, NI PXI-6513, NI PXI-6514, NI PXI-6515, NI PXI-6624, NI PXI-6704, NI PXI-8464/2, NI PXI-8231 ja NI PXI-8433/2 ovat kaikki suoraan PXI-kehikkoon liitettyjä I/O-moduuleja. Lisäksi kehikkoon on kytketty Pickeringin neljä multiplekseriä.

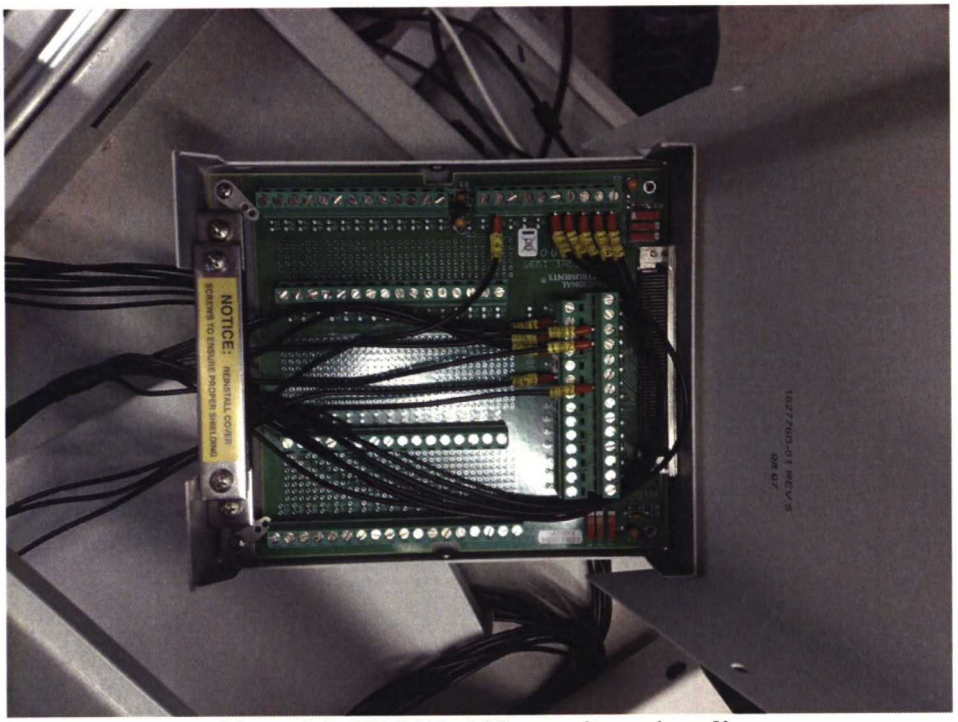
PXI-kehikon I/O-korteista tehtiin kytkennät multipleksereihin siten, että jokaisesta I/O-kortista lähtee kaapeli SH100-100-F tai SH68-68-D1 ruuviterminaalille NI SCB, joka on esitetty kuvassa 24. NI SCB-100-ruuviterminaalista on vedetty johdotukset normaalilla 0,5 neliömillimetrin sähköjohdolla Pickeringin ruuviterminaaleihin, jotka on kytketty kaapelilla 40-971-200-2M-multipleksereihin. Pickeringin ruuviterminaalit on esitetty kuvassa 23 yhtenä laatikkona, mutta todellisuudessa terminaaleja on 16 kappaletta eli 4 per multiplekseri. Kuvassa 25 on esitetty Pickeringin ruuviterminaaleja. Ohjelmoitavia jännitelähteitä ohjataan kontrollerilla GPIB-liitynnän kautta, johon lähteet on liitetty GPIB-kaapelilla. GPIB (general purpose interface bus) on erityisesti tietokoneiden, lisälaitteiden ja laborato-rioinstrumenttien väliseen kommunikaatioon suunniteltu rinnakkaismuotoinen dataväylä. Lisätietoja GPIB-väylästä ja sen toiminnasta löytyy standardista IEEE standard 488.1-1987 Standard Digital Interface for Programmable Instrumentation.

Multipleksereissä on yhteensä 80 lähtöä, joista 69 on käytetty signaalien välittämiseksi ohjausyksikköön. Yhteen tällaiseen lähtöön on ohjelmallisesti mahdollista valita multiplekseriä ohjaamalla mikä tahansa kuvan 23 multiplekseriin merkityistä signaaleista. Kuvassa on piirrettynä yksinkertaisuuden vuoksi ainoastaan yksi multiplekseri, mutta todellisuudessa multipleksereitä on neljä, joiden valittavat signaalit ovat identtisiä. Kuvassa 26 on esitetty multiplekserien lähtöjen kytkentä ohjausyksikköön AMPSEAL23-liittimillä. Kuvan pieni liitin on AMPSEAL8, jolla kytketään CAN-liitynnät ja käyttöjännite sekä maa ohjausyksikköön. Ohjausyksikön CAN-liitynnät ja käyttöjännite sekä maa sijaitsevat erillisessä liittimessä, joten niitä ei ollut tarpeellista johdottaa kulkemaan multiplekserien kautta. Testausjärjestelmän tarjoamat liitynnät RS-232, RS-485, Ethernet ja USB ovat optioliityntöjä, joita varten joudutaan aina tekemään erillinen kaapelointi, joka riippuu testauksen kohteena olevasta ohjausyksiköstä tapauskohtaisesti. Laitteistossa on kaksi

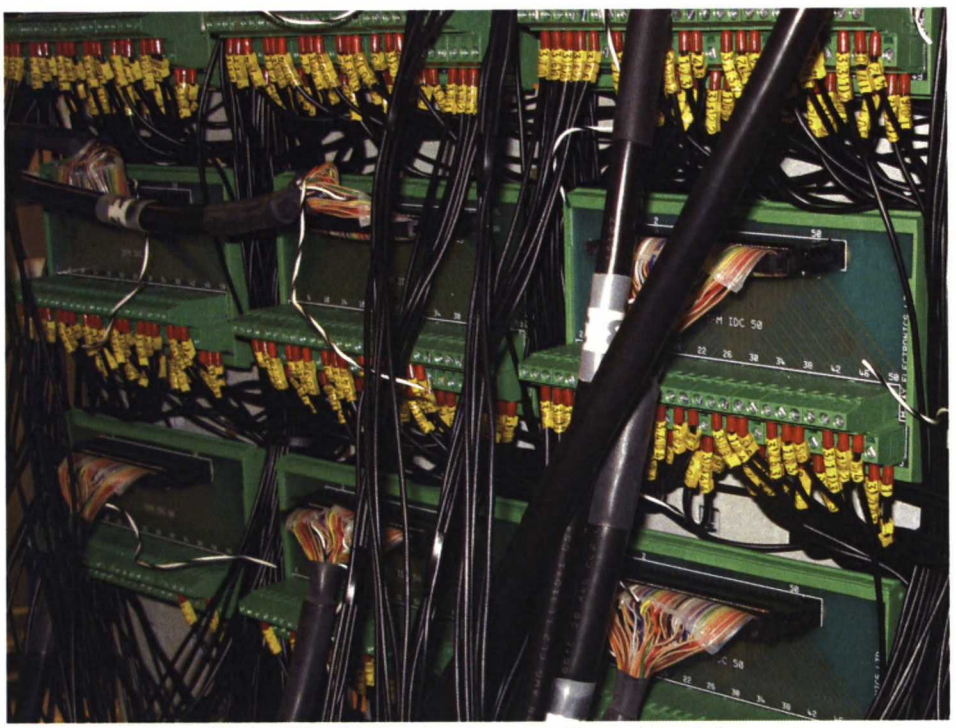
Ethernet-liityntää, jotta laitteisto voidaan kytkeä samanaikaisesti sekä ohjausyksikön mahdolliseen Ethernet-liityntään että lähiverkkoon.



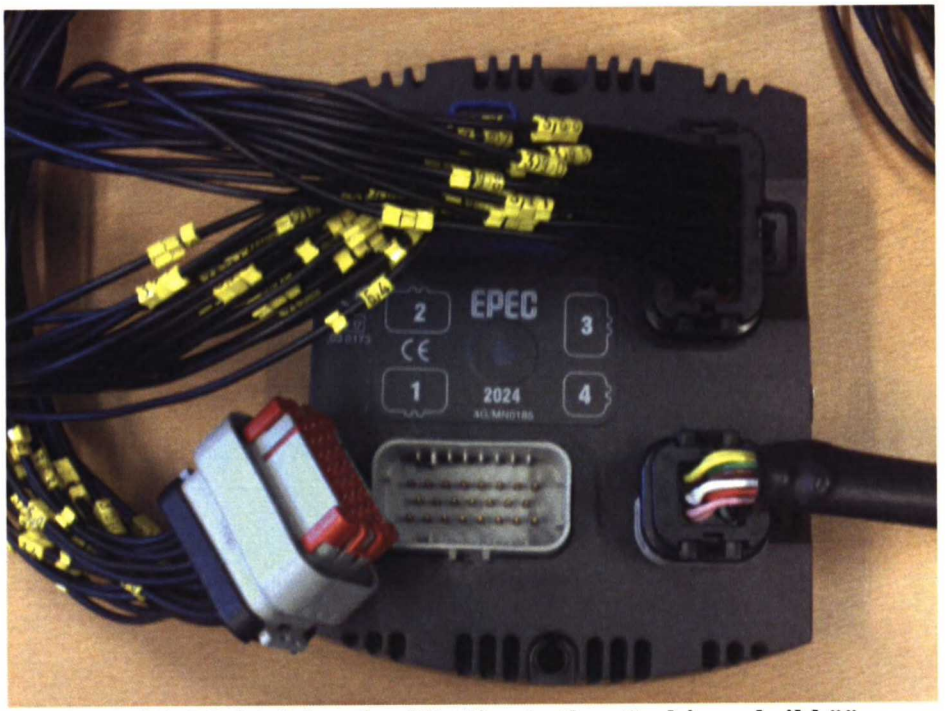
Kuva 23: Testausjärjestelmän liitynnät.



Kuva 24: NI SCB-100 ruuviterminaali.



Kuva 25: Pickeringin ruuviterminaaleja.



Kuva 26: Multiplekserien lähtöjen kytkentä ohjausyksikköön.

5.1.3. Ohjelmointi

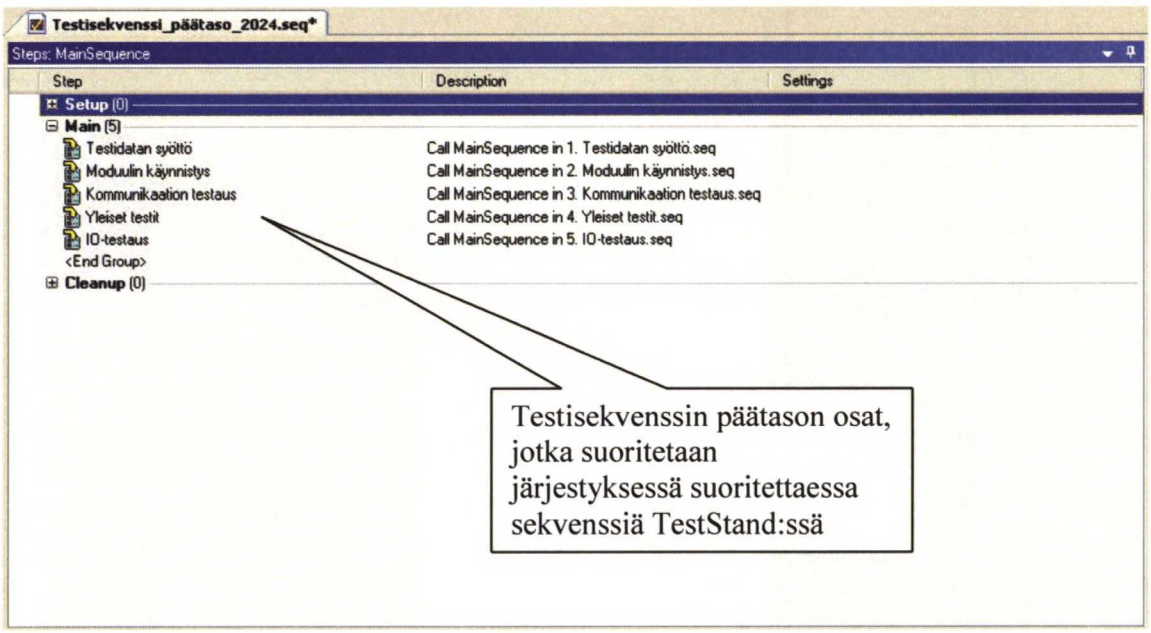
Testausjärjestelmän ohjelmamoduulit eli järjestelmän yksittäisiä toimintoja ohjaavat ohjelman palaset, joissa on toteutettu yksittäiset testit, ohjelmoitiin luvussa 4.2.1. esitellyllä LabVIEW-sovelluskehitysympäristöllä. Automaattinen testaussekvenssi, joka suorittaa LabVIEW:llä ohjelmoidut yksittäiset testit, ohjelmoitiin testien hallintaan tarkoitetulla TestStand-ohjelmistolla, joka on esitelty luvussa 4.2.2.

Seuraavassa on esitetty testaussekvenssi, jonka mukaan TestStand suorittaa LabVIEW:llä ohjelmoidut yksittäiset testit järjestyksessä:

- Testisekvenssi:
    1. Alkutoimenpiteet
    2. Testidatan syöttö
    3. Ohjausyksikön käynnistys
    4. Kommunikaation testaus
    5. Yleiset testit
    6. Ohjausyksikön I/O-testaus
    7. Raportointi

Kuvassa 27 on esitetty testaussekvenssin päätaso TestStand:ssä. Kuvasta voi huomata, että se sisältää kaikki samat kohdat kuin edellä esitelty testisekvenssi, vaikka nimet eivät ole

täsmälleen samat. Alkutoimenpiteet ja raportointi eivät kuitenkaan näy TestStand:in sekvenssissä.



Kuva 27: Testisekvenssin päätaso.

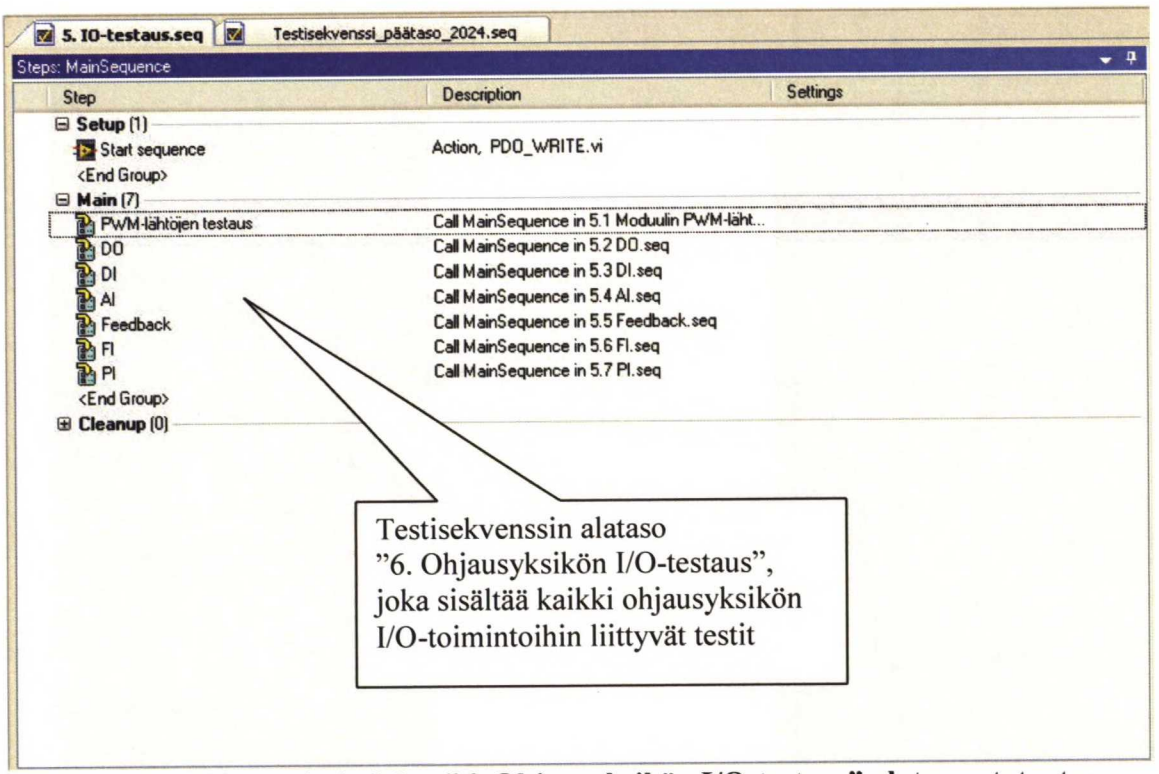
Testisekvenssin päätaso sisältää kaikki testausprosessin vaiheet alkutoimenpiteistä testitapahtuman raportointiin. Sekvenssi on toteutettu siten, että sen päätasolta voidaan valita ajettavaksi kaikki osat tai jokin yksittäinen kohta erikseen, jolloin kaikkia sekvenssin osia ei tarvitse välttämättä ajaa jokaisella ajokerralla. Jokainen ohjausyksiköstä testattava toiminto kuuluu johonkin näistä testaussekvenssin päätason osista. Myös alemmille tasoille toteutetut yksittäiset testit voidaan ajaa yksittäin, jos ei ole tarvetta ajaa kaikkia sekvenssiin kuuluvia testejä.

Testisekvenssin päätasot on esitelty tarkemmin seuraavassa:

1. Alkutoimenpiteet
   - Alkutoimenpiteet suoritetaan ensimmäiseksi automaattista testausta aloitettaessa. Tässä vaiheessa alkutoimenpiteisiin kuuluu manuaalisia tehtäviä, kuten sovelluksen ja laiteohjelmiston lataus ohjausyksikköön. Tämä osa sekvenssistä on tarkoitus toteuttaa myöhemmin automaattisesti niin, että manuaaliset tehtävät poistuvat.
2. Testidatan syöttö
   - Tässä kohdassa identifioidaan testattavan ohjausyksikön malli ja käytetyn testaussovelluksen versio sekä valitaan halutaanko ajaa kaikki testit vai vain jokin osa testeistä.

3. Ohjausyksikön käynnistys
   - Ohjausyksikön käynnistyksessä suoritetaan ohjausyksikön hallittu käynnistys sekä käynnistykseen liittyvät tarkistukset, kuten esimerkiksi jännitteen syöttö ohjausyksikköön ja sovelluksen oikeellinen käynnistyminen.
4. Kommunikaation testaus
   - Kommunikaation testauksen yhteydessä testataan ohjausyksikön kaikki kommunikaation toimintaan liittyvät toiminnot, kuten esimerkiksi CANopen-protokollan toiminta ja kaikkien ohjausyksikön sisältämien kommunikaatiovaihtoehtojen, mahdollisesti esimerkiksi RS-232:n, toiminta.
5. Yleiset testit
   - Yleiset testit kohdassa suoritetaan kaikki sellaiset testit, joita ei voi identifioida kuuluvaksi minkään muun sekvenssin päätason otsikon alle. Tässä kohdassa suoritetaan muun muassa Flash-muistin testaus.
6. Ohjausyksikön I/O-testaus
   - Tässä kohdassa suoritetaan kaikki testattavan ohjausyksikön I/O-toimintoihin liittyvät testit. I/O-toimintoja voivat olla muun muassa pulssinleveysmoduloitujen lähtöjen ohjaus ja digitaalisten liityntöjen toiminta moduulin mallista riippuen.
7. Raportointi
   - Kaikkien osatestien suorituksen jälkeen täytyy testauksen vaiheista ja tuloksista koostaa helposti ymmärrettävä ja kaiken tarpeellisen testidatan sisältävä testausraportti. Testausraportti koostuu TestStand:ssä taustalla ajettaessa testisekvenssiä, mutta tässä kohdassa koostetaan lopullinen raportti ja esitetään se testaajalle sekä tallennetaan määriteltyyn paikkaan, johon kaikki testausjärjestelmän tuottamat testausraportit kerätään myöhempää tarkastelua varten.

Testisekvenssin toteutus on kaikille testiobjekteille eli erilaisille ohjausyksiköille samanlainen. Jokaiselle testiobjektille on määritelty I/O-kartta, jonka avulla testausjärjestelmä identifioi tietyn ohjausyksikkötyypin I/O-pinnien toiminnot testausta varten. Testiobjektien ainoat eroavaisuudet ovatkin juuri I/O-rajapinnassa sekä kommunikaatiovaihtoehdoissa. Kun ohjausyksikön I/O-rajapinta on tunnistettu I/O-kartan avulla, testausjärjestelmä tietää mihin ohjausyksikön pinniin täytyy tiettyjä testejä kohdistaa. LabVIEW:llä ohjelmoidut ohjelmamoduulit saavat siis I/O-kartan avulla kaiken tarvitsemansa tiedon tietyn moduulityypin I/O-testausta varten. Näin ohjelmamoduulien sisäinen toteutus on voitu tehdä siten, että erilaisille ohjausyksikkötyypeille voidaan käyttää samoja ohjelmamoduuleja. Ainoastaan ohjelmamoduulin parametrit vaihtelevat ohjausyksikkötyypin mukaan. Kuvassa 28 on esitetty testausjärjestelmän testaussekvenssin kohdan 6. (Ohjausyksikön I/O-testaus) alatason toteutus.



Kuva 28: Sekvenssin kohdan ”6. Ohjausyksikön I/O-testaus” alatason toteutus.

5.1.4. Testaus

Testausjärjestelmän toteutusvaiheessa järjestelmälle suoritettiin neljä erillistä testausvaihetta laitteiston ja toimintojen oikean toiminnan varmistamiseksi. Nämä testausvaiheet olivat:

1. Laitteiston toiminnallinen testaus
2. Ohjelmamoduulien yksikkötestit
3. Ohjelmiston integraatiotestaus
4. Koko järjestelmän toiminnallinen testaus

Ensimmäinen testausvaihe oli testausjärjestelmän laitteiston toiminnallinen testaus, jossa testattiin laitteiston kaikkien yksittäisten komponenttien toiminta. Lisäksi testattiin kaikki johdotukset ja kytkennät, jotta voitiin varmistua, että kaikki signaalit saadaan kytkettyä haluttuihin paikkoihin. Tässä vaiheessa luotiin LabVIEW:llä erilliset manuaaliseen testaukseen soveltuvat ohjelmamoduulit jokaisen testausjärjestelmän toiminnon testaukseen. Käytännössä tämä tarkoittaa, että jokainen multiplekserin lähtö, joka voidaan kytkeä ohjausyksikön pinniin, käytiin läpi valitsemalla multiplekserin lähtöön vuorotellen kaikki siihen mahdollisesti kytkettävät signaalit ja mitattiin ne yleismittarilla tai oskilloskoopilla.

Tämän testausvaiheen avulla varmistuttiin kytkentöjen oikeellisuudesta ja yksittäisten korttien toiminnasta.

Toisessa testausvaiheessa toteutettiin LabVIEW:llä toteutettujen ohjelmamoduulien yksikkötestit. Ohjelmamoduulit ovat testisekvenssin osia eli käytännössä sekvenssin osia, jotka suorittavat yksittäiset testit. Tässä vaiheessa testattiin ohjelmamoduulien määrittelyn mukainen toiminta, jotta voitiin varmistua yksittäisten testien oikeellisesta toiminnasta. Testit toteutettiin ohjelmamoduulien ohjelmoinnin yhteydessä seuraamalla ohjelmamoduulien ohjaamien korttien toimintaa ja CAN-väyläliikennettä.

Kolmas testausvaihe testausjärjestelmän toteutuksen aikana oli ohjelmiston integraatiotestaus. Integraatiotestauksen tarkoitus oli varmistaa käytännössä testisekvenssin toiminta. Tämä testausvaihe suoritettiin ainoastaan havaintopohjalta ja tarkkailemalla testisekvenssin tuottamaa testiraporttia, koska testisekvenssi on toteutettu siten, että sekvenssin osien välillä ei ole varsinaisia riippuvaisuuksia.

Neljäs ja viimeinen testausvaihe järjestelmän testauksessa oli koko testausjärjestelmän toiminnallinen testaus. Tässä vaiheessa ohjelmoitiin testiobjektille eli ohjausyksikölle testattava laiteohjelmisto, johon oli tarkoituksella luotu tiettyjä virheitä ja virheellisiä toiminnallisuuksia. Tätä virheellistä laiteohjelmistoa testattiin sitten testausjärjestelmällä, jolloin saatiin paljon hyödyllistä informaatiota järjestelmän kyvystä löytää virheitä sekä tietoa järjestelmän toiminnasta tilanteissa, jolloin laiteohjelmistosta löytyy virhe.

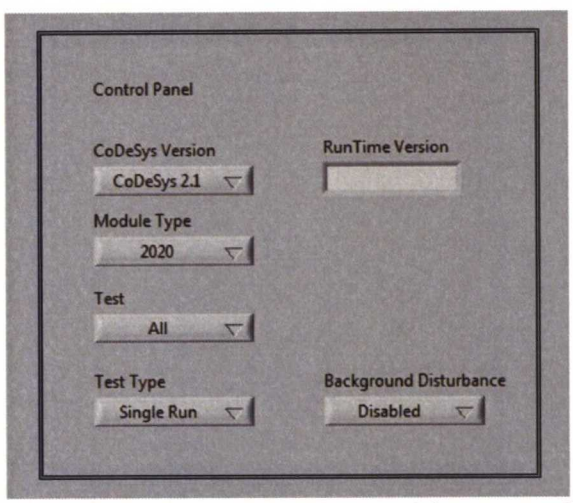
5.2. Testausjärjestelmän käyttö

Testauksen kannalta testausjärjestelmä on hyvin yksinkertainen käyttää, johtuen sen automaattisuudesta. Tässä vaiheessa aivan kaikkia toimintoja, kuten sovelluksen ja laiteohjelmiston latausta ohjausyksikköön, ei ole automatisoitu, mutta myös nämä toiminnot tullaan automatisoimaan jatkokehitysprojektien aikana. Testausjärjestelmän jatkokehitys ei kuulu tämän työn alueeseen. Testausjärjestelmää käyttävän testaajan ei tarvitse, ohjausyksikön ohjelmien ja asetusten ollessa kunnossa, tehdä muuta kuin syöttää järjestelmälle sen tarvitsemat tiedot testausobjektista ja kytkeä testattava ohjausyksikkö järjestelmään, kuten kuvassa 26 on aiemmin esitetty. Testausjärjestelmä pyytää ennen testisekvenssin testausosuuden ajamista TestStand:llä testaajalta seuraavat tiedot LabVIEW:llä ohjelmoidun ohjauspaneelin kautta:

- Pudotusvalikko: CoDeSys Version
  - Tästä valikosta valitaan aiemmin luvussa 3.2.3. tarkemmin esitellyn CoDeSys-sovelluskehitysympäristön versio. Kehitysympäristön versio vaikuttaa tiettyihin testisekvenssin osiin ja niissä ajettaviin ohjelmamoduuleihin, koska kehitysympäristön eri versioiden, tällä hetkellä 2.1 ja 2.3, tuottamat testaukseen käytetyt sovellukset eivät ole identtisiä, johtuen niitä vastaavien laiteohjelmistojen eroavaisuuksista.

- Pudotusvalikko: Module Type
  - Tästä valikosta valitaan testiobjektin eli ohjausyksikön malli, esimerkiksi 2024, joka on esitetty kuvassa 3 sivulla 8. Ohjausyksikön malli vaikuttaa ohjelmamoduulien parametreihin, koska eri malleilla on erilaiset I/O-liitynnät.
- Pudotusvalikko: Test
  - Tällä valikolla valitaan ajettavat testit. Testejä voidaan ajaa yksittäin, ryhmissä tai kaikki testit kerrallaan. Testiryhmään voidaan valita halutut ominaisuudet testattavaksi, mikä tarkoittaa, että sekvenssistä ajetaan vain valitut osat.
- Pudotusvalikko: Test Type
  - Tällä pudotusvalikolla valitaan ajettavien testien tyyppi eli ajetaanko valitut testit kerran läpi vai toistetaanko testejä silmukassa useita kertoja.
- Syötelaatikko: RunTime Version
  - Tämän valikon avulla määritellään laiteohjelmiston versio. Laiteohjelmiston versio merkitään myöhemmin testausraporttiin ja versiota käytetään myös testisekvenssin identifiointiin. Laiteohjelmiston eri versioille saattaa olla erilainen testisekvenssi, johtuen versioiden välisistä eroavaisuuksista.
- Pudotusvalikko: Background Disturbance
  - Tämän valikon avulla voidaan testaussekvenssiin lisätä taustahäiriöitä, kuten taustaliikennettä CAN-väylään tai esimerkiksi erilaisia kuormituksia muihin toimintoihin testattaessa jotain tiettyä toimintoa.

Kuvassa 29 on esitelty testisekvenssin käyttämä ohjauspaneeli, jonka avulla testausjärjestelmä identifioi testiobjektin. Testisekvenssi esittää ohjauspaneelin ponnahdusikkunana ennen varsinaisen sekvenssin testausosion alkamista. Pyydettyjen tietojen avulla sekvenssin yksittäiset testit saavat tarvittavat tiedot testattavan ohjausyksikön testien ajamiseen.



Kuva 29: Testausjärjestelmän ohjauspaneeli.

Kun ohjauspaneelissa kysytyt tiedot on syötetty järjestelmään, ei testaajalta vaadita muita toimenpiteitä testisekvenssin suorituksessa. Automaattinen testisekvenssi suoriutuu läpi automaattisesti, minkä jälkeen TestStand generoi testausraportin ajettujen testien tuloksista. Testaajan tärkein tehtävä onkin testitulosten oikea ja järkevä tulkinta, jotta testausjärjestelmästä olisi hyötyä ohjausyksiköiden testauksessa.

5.2.1. Testisekvenssin suoritus

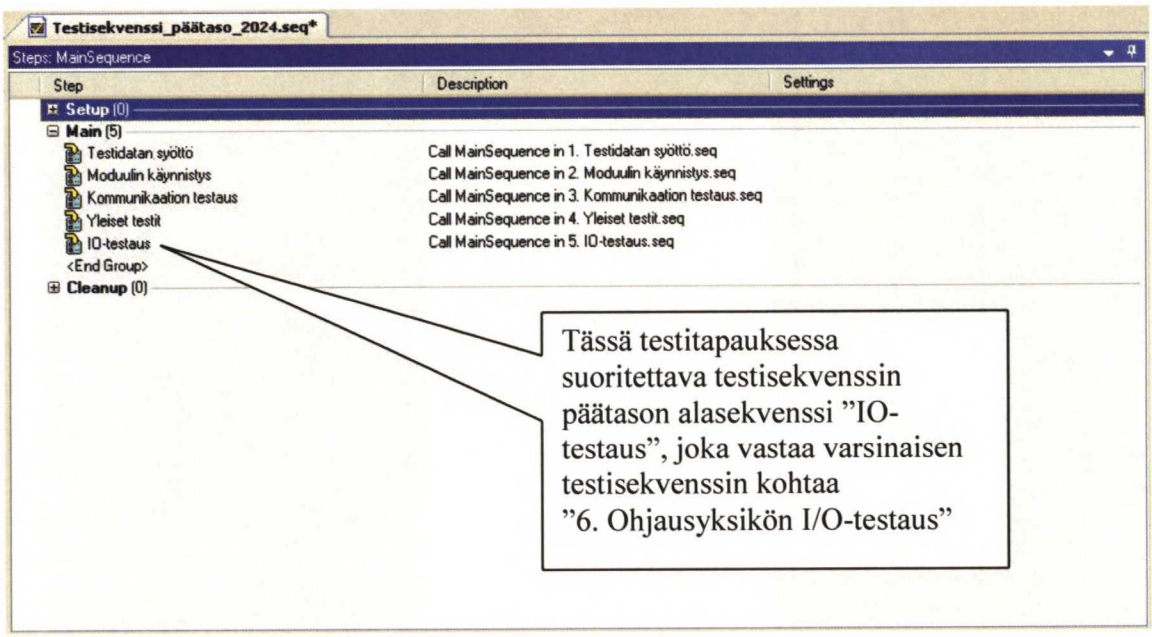
Tässä kappaleessa käydään läpi yksityiskohtaisemmin sivuilla 50 ja 51 esitellyn testisekvenssin päätason kohdan "6. Ohjausyksikön I/O-testaus" alasekvenssin "6.1 Moduulin PWM-lähtöjen testaus" suorittaminen. Koko testisekvenssin suoritus seuraa samaa kaavaa yhden sekvenssin osakokonaisuuden suorituksen kanssa. Koko sekvenssin kuvaus tässä kappaleessa ei sekvenssin pituudesta johtuen kuitenkaan ole mielekästä, joten esittelyyn on valittu ainoastaan yksi kohta testisekvenssistä. Kuvissa esitettyjen testisekvenssin tasojen numerot eroavat varsinaisen testisekvenssin kuvauksen numeroinnista, koska kaikkia testisekvenssin osia, kuten alkutoimenpiteet, ei suoriteta testausjärjestelmällä.

Alasekvenssi 6.1 testaa nimensä mukaisesti ohjausyksikön PWM-lähtöjen toimintaa. Kuvassa 30 on esitetty testisekvenssin kohdan "6.1 Moduulin PWM-lähtöjen testaus" suoritettavien testien kuvaus. Kuvassa on esitetty testausjärjestelmän suorittamat toiminnot, ohjausyksikön suorittamat toiminnot sekä testausjärjestelmän ja ohjausyksikön välinen kommunikaatio CAN-väylän avulla käyttäen CANopen-protokollan mukaisia viestejä.

Testeri:	CAN-viestit:	Moduuli (CoDeSys):
6.1 Moduulin PWM-lähtöjen testaus:	Liipaisu: 202, 8, 06, 01, 01, 00, 00, 00, 32, 00 ->	
PWM lähtöjen pitäisi asettua 140 Hz:aan ilman erillistä asetusta.		Annetaan CoDeSysissä PWM-lähdöille numeerinen arvo 16384. (=Pulssisuhde 50%)
Mitataan lähtöjen taajuus.		
Asetetaan kaikki PWM-lähdöt tajuuteen 2550 Hz pulssisuhteella 50%.	Liipaisu: 202, 8, 06, 01, 02, 00, 00, 00, 32, ff ->	CoDeSysissä PWM-lähtöjen asetus.
Mitataan lähtöjen taajuus.		
Asetetaan kaikki PWM-lähdöt tajuuteen 10 Hz pulssisuhteella 50%.	Liipaisu: 202, 8, 06, 01, 02, 00, 00, 00, 32, 01 ->	CoDeSysissä PWM-lähtöjen asetus.
Mitataan lähtöjen taajuus ja kirjataan nämä arvot ylös raporttiin.		
Asetetaan kaikki PWM-lähdöt tajuuteen 0 Hz pulssisuhteella 0%.	Liipaisu: 202, 8, 06, 01, 02, 00, 00, 00, 00, 00 ->	Annetaan CoDeSysissä PWM-lähdöille numeerinen arvo 0.
Viestien RPDO1 ja TPDO2 nollaus.	Liipaisu: 202, 8, 00, 00, 00, 00, 00, 00, 00, 00 ->	

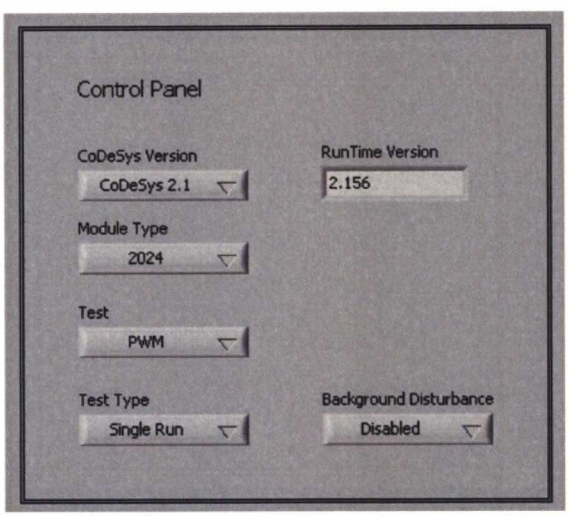
Kuva 30: Alasekvenssin "6.1 Moduulin PWM-lähtöjen testaus" testikuvaus.

Ennen varsinaista sekvenssin suoritusta TestStand:llä, täytyy ohjausyksikkö kytkeä asianmukaisesti testausjärjestelmään. Kun ohjausyksikkö on kytketty testausjärjestelmään ja TestStand:iin on avattu testisekvenssin päätaso, joka on esitetty kuvassa 31, voidaan testisekvenssin suoritus aloittaa. Ennen edellä kuvailtuja toimenpiteitä ohjausyksikkö tulee valmistella siten, että siihen oli ladattu testattava laiteohjelmiston versio sekä testaukseen tarkoitettu sovellusohjelma. Myös ohjausyksikön kommunikaatioparametrit tulee olla aseteltuna testausjärjestelmän vaatimalla tavalla testisekvenssin aluksi (Ohjausyksikön solmunumero = 1, ohjausyksikön sisäinen päätevastus päällä, ohjausyksikkö asetettuna CANopen-isännäksi ja kommunikointinopeus = 250 kbit/s).



Kuva 31: Testisekvenssin päätaso.

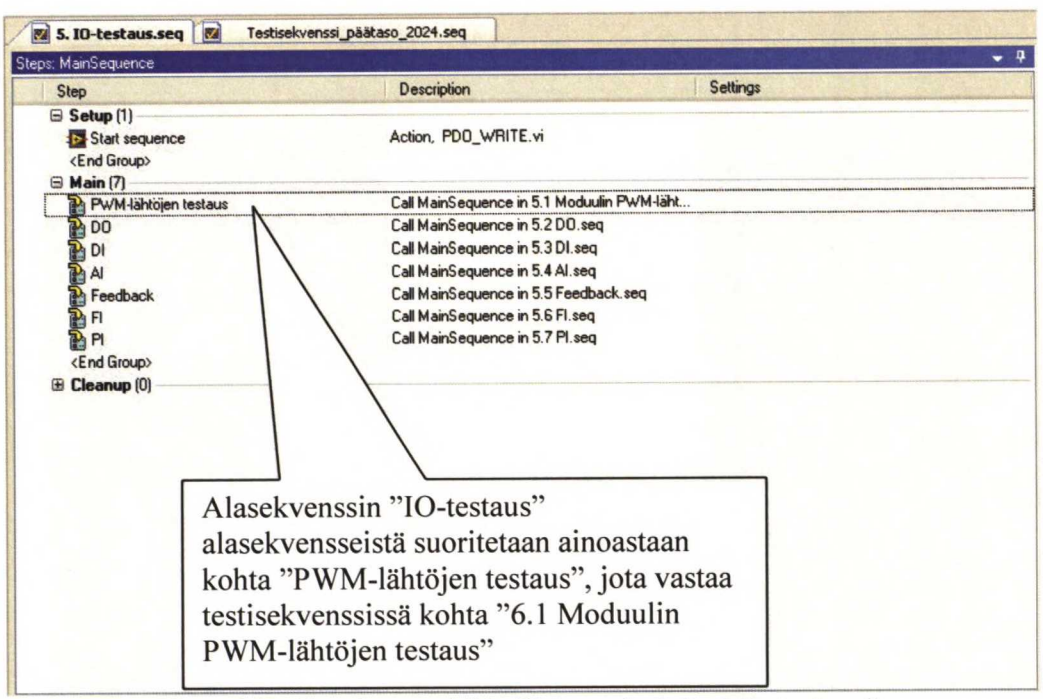
Testisekvenssin käynnistyksen jälkeen ensimmäinen suoritettava vaihe oli "Testidatan syöttö". Tässä kohdassa syötettiin testausjärjestelmän vaatimat esitiedot testiobjektista testisekvenssin suorittamaan ohjauspaneeliin kuvan 32 mukaisesti. Ohjauspaneeliin syötettyjen tietojen perusteella testausjärjestelmä osaa identifioida testiobjektin sekä saa tarpeellisen informaation ajettavista testeistä. Tässä tapauksessa haluttiin ajaa ainoastaan PWM-lähtöjen testiosio Epec 2024-ohjausyksikölle, jossa oli CoDeSys 2.1-versiolla tehty sovellusohjelma. Laiteohjelmiston versio oli 2.156 ja taustahäiriöitä ei testauksessa haluttu käyttää. Testin tyypiksi valittiin "Single Run", joka tarkoittaa että testi ajettiin ainoastaan kerran läpi.



Kuva 32: Ohjauspaneeliin syötetty testidata.

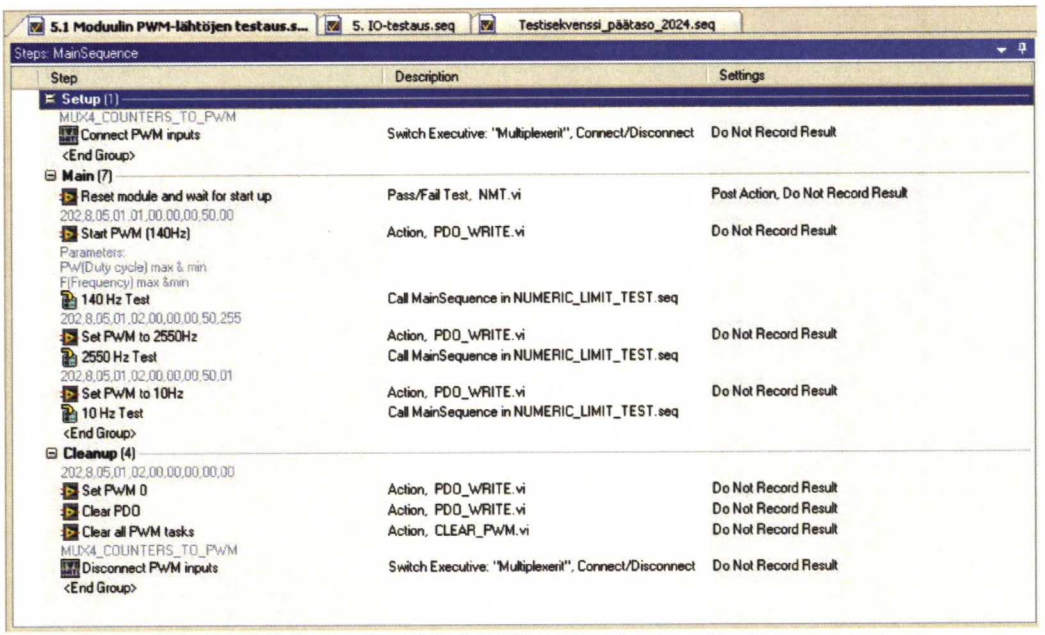
Ohjauspaneeliin syötetyn testidatan mukaisesti sekvenssi lähti etenemään testisekvenssin päätasolla. Kuvassa 31 testisekvenssin päätason toinen kohta on "Moduulin käynnistys". Tämä kohta on alasekvenssi, josta löytyy LabVIEW:llä ohjelmoitu ohjelmamoduuli, joka asettaa ohjausyksikön käyttöjännitteen +24 volttiin. Alasekvenssistä "Moduulin käynnistys" löytyy myös toinen LabVIEW:llä ohjelmoitu ohjelmamoduuli, joka lukee Excel-tiedostosta Epec 2024-ohjausyksikön I/O-kartan. I/O-kartan perusteella testausjärjestelmä tietää missä ohjausyksikön pinneissä PWM-lähdöt sijaitsevat. Myös muissa I/O-testeissä käytetään moduulin käynnistysvaiheessa luettua I/O-karttaa hyväksi, jotta tiedetään missä ohjausyksikön pinneissä testin kohteet sijaitsevat. I/O-karttaa muuttamalla testiobjekteiksi voidaan helposti lisätä uusia ohjausyksikkömalleja myöhemmin.

Edellä kuvailtujen suoritusten jälkeen sekvenssi hyppää suoraan kohtaan "IO-testaus", jonka alasekvenssinä on toteutettu testisekvenssin kohta "6.1 Moduulin PWM-lähtöjen testaus". Kuvassa 33 on esitetty testisekvenssin päätason alasekvenssi "6. Ohjausyksikön I/O-testaus". Testisekvenssin muita osia ei suoriteta tässä testiajossa, koska ohjauspaneelista valittiin ajettavaksi testiksi ainoastaan PWM-testaus, jolloin TestStand osaa jättää siihen ohjelmoitujen ehtojen avulla muut testit suorittamatta.



Kuva 33: Alasekvenssi "6. Ohjausyksikön I/O-testaus".

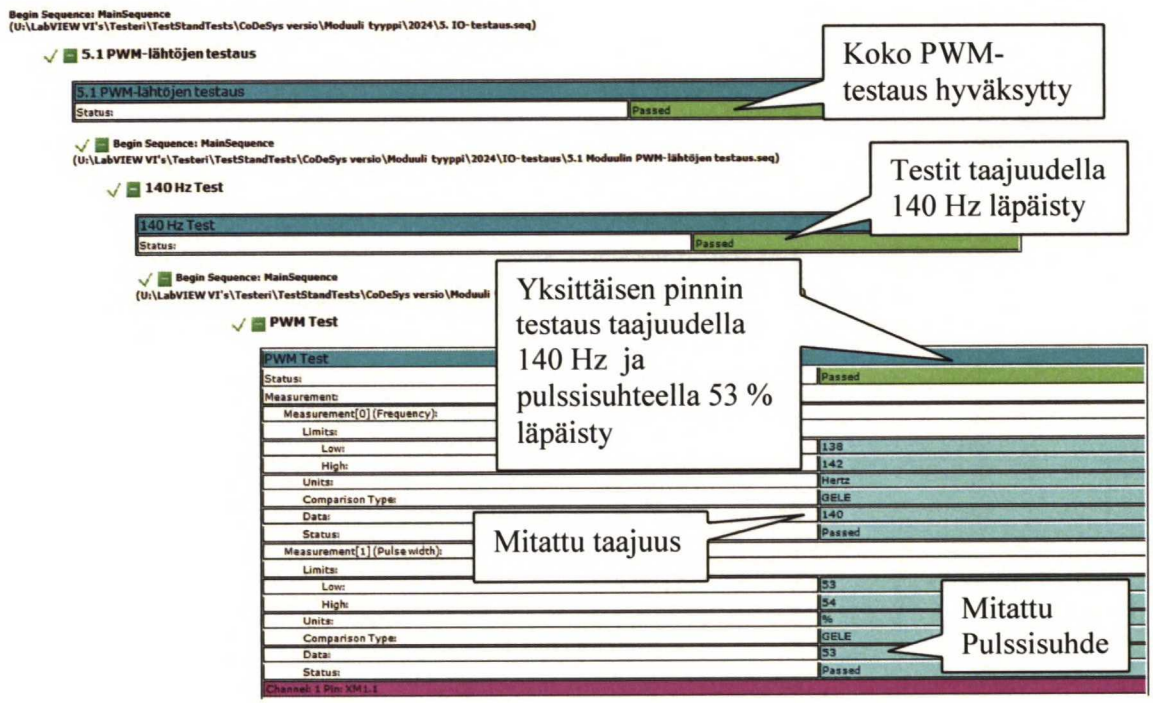
Kuvassa 34 on esitetty alasekvenssin "6.1 Moduulin PWM-lähtöjen testaus" alatason toteutus TestStand:ssä, mistä löytyy kaikki yksittäiset testejä ohjaavat LabVIEW:llä toteutetut ohjelmamoduulit.



Kuva 34: "6.1 Moduulin PWM-lähtöjen testaus" alatason toteutus.

Kuvassa 34 esitetty PWM-lähtöjen testaus seuraa suoraan kuvassa 30 esitettyä testikuvausta. Kuvan esittämä testaussekvenssi alkaa PWM-mittauskortin NI PXI-6624 mittauspinnien kytkennällä multiplekserien avulla ohjausyksikön PWM-lähtöpinneihin. Epec 2024-ohjausyksikön I/O-kartta, joka luettiin testisekvenssin päätason alussa, kertoo sekvenssille, mihin ohjausyksikön pinneihin PWM-mittaukset kytketään. Tämän jälkeen suoritetaan kuvassa 30 esitetyt testitoimenpiteet ja lopuksi PWM-mittauskytkennät poistetaan multipleksereiltä.

Testisekvenssin suorituksen jälkeen TestStand koostaa ajettujen testien vaiheista ja tuloksista testausraportin, joka on esitetty kuvassa 35. Kuvassa on esitetty osa kokonaisesta testausraportista, koska raportti on pitkä, johtuen testattavien pinnien lukumäärästä ja niiden testauksesta tallennettavasta testidatasta. Kuvassa ylimpänä näkyy, että koko testisekvenssin osa "6.1 Moduulin PWM-lähtöjen testaus" on suoritettu hyväksytysti. Seuraavalla tasolla ilmaistaan, että testit taajuudella 140 Hz on suoritettu kaikille ohjausyksikön PWM-lähdöille hyväksytysti. Alimmalla tasolla näkyy avattuna yksittäisen pinnin, tässä tapauksessa liittimen yksi pinnin yksi (XM1.1), testitulos ja pinnin testauksesta kerätty testidata. Yksittäisen pinni testidatassa näkyy olennaisimpina tietoina, että se on testattu hyväksytysti ja mitattu taajuus on ollut tasan 140 Hz ja pulssisuhde 53 % niin kuin pitikin olla. Muista testisekvenssin kohdista löytyy samanlaiset kuvaukset testiraportista taajuuksittain ja pinneittäin. Samankaltainen testiraportti luodaan myös ajettaessa muita testisekvenssin testejä. Testiraporttiin haluttu testidata on määrätty testikohtaisesti niin, että kaikki mahdollisesti tarpeellinen informaatio tulisi raporttiin.



Kuva 35: Testausraportti.

5.3. Testausjärjestelmän ylläpito

Testausjärjestelmän ylläpito ja ylläpidettävyys ovat tärkeitä asioita, jotka tulee ottaa huomioon jo järjestelmän suunnitteluvaiheessa. Järjestelmän ylläpidettävyys vaikuttaa suoraan järjestelmän elinkaaren pituuteen. Helposti ylläpidettävä järjestelmä auttaa myös olennaisesti testausprosessin jatkokehityksessä tulevaisuudessa. Helposti ylläpidettävään järjestelmään voidaan vaivattomasti ja nopeasti lisätä uusia laitteita ja testejä. Erilaisten testitapausten helppo ja nopea lisääminen onkin olennainen osa järjestelmän sujuvaa käyttöä. Laitteiston osalta hyvä ylläpidettävyys merkitsee laajennuksien helppoa toteuttamista. Kun laitteistoa halutaan muuttaa tai laajentaa soveltumaan uusiin testikohteisiin, täytyy sen olla nopeaa ja vaivatonta, jotta muutos tai laajennus olisi mielekästä verrattaessa uuden järjestelmän toteuttamiseen.

Testausjärjestelmän elinkaari määrää koko järjestelmän kehittämisen mielekkyyden. Jo järjestelmää suunniteltaessa täytyy elinkaari ottaa huomioon, jotta järjestelmän kustannukset saataisiin katettua parantuneena ja nopeutuneena testausprosessina, jota voidaan toistaa jopa tuhansia kertoja erilaisille ohjausyksiköille. Kallista järjestelmää tulee voida käyttää tehokkaasti testaukseen useiden vuosien ajan, jotta sen rakentaminen olisi tämän työn laajuudessa yleensäkään mielekästä. Tärkeimpiä yksittäisiä vaikuttimia järjestelmän elinkaareen onkin juuri sen ylläpidettävyys. Testausjärjestelmän tärkein tehtävä on tuottaa elinkaarensa aikana mahdollisimman paljon lisäarvoa ohjausyksiköiden suunnitteluprosessiin

parantamalla, nopeuttamalla ja muutenkin kaikin tavoin edesauttamalla ohjausyksiköiden suunnitteluprosessin testausvaiheita.

5.3.1. Ylläpidettävyys

Testausjärjestelmän ylläpidettävyyden maksimoimiseksi on järjestelmän suunnittelussa, rakennuksessa ja ohjelmoinnissa otettu huomioon monia ylläpidettävyyteen vaikuttavia asioita, kuten laitteiston laajennettavuus ja ohjelmiston uudelleenkäytettävyys. Jotta järjestelmän ylläpidettävyys saataisiin maksimoitua, tulee järjestelmän olla mahdollisimman riippumaton muutoksista, jotka kohdistuvat testausobjektiin, testausobjektin ja järjestelmän rajapintoihin tai käytettyihin testeihin. Tässä työssä toteutettuja tärkeitä ratkaisuja testausjärjestelmän toteutuksessa ylläpidettävyyden näkökulmasta ovat muun muassa PXI-arkkitehtuurin käyttö järjestelmän perustana, testausjärjestelmän ja testausobjektin rajapinnan yleisyys sekä testimoduulien yleiskäyttöisyys.

Testausjärjestelmän ylläpitoon vaikuttavista tekijöistä tärkeimpänä voisi varmasti pitää PXI-arkkitehtuuria, joka mahdollistaa modulaarisuutensa ansiosta vaivattoman ja nopean laitteiston laajennuksen tarvittaessa. PXI-arkkitehtuuri mahdollistaa laitteiston laajennuksen samanlaisilla korteilla, kun mitä järjestelmästä jo löytyy, siten että samoja LabVIEW-ohjelmia voidaan suoraan uudelleen käyttää laajennuskorttien ohjauksessa. Luvussa 4.3.2. on esitelty tarkemmin PXI-arkkitehtuuria ja sen ominaisuuksia.

Koska jokaisessa testausjärjestelmän testausobjektissa eli testattavassa ohjausyksikössä on käytössä samanlainen liitintekniikka, voitiin rajapinta järjestelmän ja testiobjektin välille rakentaa helposti yleiskäyttöiseksi siten, että jokaiseen rajapinnan liittimen pinniin on mahdollista kytkeä kaikki testausjärjestelmän tarjoamat I/O-signaalit multiplekserien kautta. Tämä toteutustapa parantaa järjestelmän ylläpidettävyyttä huomattavasti, koska erillisiä liityntöjen muutoksia ei testiobjektin vaihtuessa tarvita. Lisäksi ohjausyksiköt, jotka sisältävät erilaisia I/O-signaaleja vastinkohdissaan, voidaan kytkeä samalla AMPSEAL23-liittimellä testausjärjestelmään, ohjelmiston hoitaessa oikeat signaalit oikeisiin pinneihin ohjausyksiköstä riippuen.

Myös LabVIEW:llä ohjelmoidut ohjelmamoduulit on toteutettu siten, että riippumatta testiobjektista, voidaan samoja ohjelmamoduuleja käyttää uudelleen samoihin tehtäviin. Testausjärjestelmään määritellään kaikkien testiobjektien I/O-kartat erillisissä taulukoissa, joiden avulla testausjärjestelmän sekvenssi ohjaa kytkentöjä ja ohjelmamoduulien toimintaa. Toisin sanoen samoja ohjelmamoduuleja voidaan käyttää uudelleen ja uudelleen erilaisille testiobjekteille, muuttamalla ainoastaan testisekvenssin parametrit vastaamaan testiobjektin I/O-karttaa, jotta järjestelmä tietää mitä signaaleja tietty ohjausyksikön pinni voi lähettää ja/tai vastaanottaa.

5.3.2. Elinkaari

Tässä työssä toteutetussa testausjärjestelmässä ei ole komponentteja, joiden elinikä vaikuttaisi olennaisesti koko järjestelmän elinkaaren pituuteen. Jotta testausjärjestelmä maksaisi itsensä takaisin, järjestelmällä pitäisi olla mahdollista testata kolmesta kuuteen uutta laiteohjelmiston versiota, jotta se olisi tuottanut hyötyä sen rakentamiseen käytetyn vaivan verran. [22]

Testausjärjestelmällä testattava laiteohjelmisto päivittyy käytännössä koko ajan, mikä tarkoittaa, että uusia versioita saattaa tulla useinkin järjestelmän testattavaksi. Laiteohjelmistoon joudutaan tekemään tai halutaan tehdä usein muutoksia ja parannuksia. Tästä johtuen testausjärjestelmää tullaan tarvitsemaan laiteohjelmiston testaukseen ainakin yhtä kauan, kun testiobjekteja eli ohjausyksiköitä on markkinoilla tarjolla. Testiobjektien elinkaaren puitteissa tämä tarkoittaa ainakin kymmentä vuotta ja todennäköisesti paljon pidempää aikaa. Tämä ei kuitenkaan ole ongelma ainakaan laitteiston komponenttien elinkaaren puolesta. Elinkaarinäkymä tässä työssä toteutetulle testausjärjestelmälle on noin kymmenestä viiteentoista vuotta.

Testausjärjestelmän ylläpidettävyys vaikuttaa suuresti sen elinkaaren pituuteen. Se, kuinka mielekästä järjestelmää on ylläpitää ja kehittää, määrää käytännössä järjestelmän elinkaaren pituuden. Ylläpidettävyyden näkökulmasta tässä työssä toteutettua järjestelmää on helppo laajentaa, jolloin järjestelmän testiobjektien määrää voidaan lisätä käsittämään uusia tulevia tuotteita, mikä kasvattaa järjestelmän elinkaaren pituutta huomattavasti. Testauslaitteiston modulaarisuuden ja ohjelmiston hyvän uudelleenkäytettävyyden ansiosta uusien testiobjektien lisääminen järjestelmään on helppoa vaarantamatta vanhojen testiobjektien testattavuutta.

6. Työn tarkastelu

Tässä luvussa tarkastellaan työn tuloksia ja saavutuksia verraten niitä luvussa 2. asetettuihin tavoitteisiin ja työn lähtökohtiin. Luvun lopussa pohditaan lyhyesti testausjärjestelmän tulevaisuutta jatkokehitysprojekteja ja ohjausyksiköiden laiteohjelmiston testausta silmälläpitäen.

6.1. Yleistä

Tämän diplomityön tarkoitus oli koneenohjausyksikön laiteohjelmiston automaattisen testausjärjestelmän suunnittelu ja toteutus. Laiteohjelmiston automaattista testausta varten työssä suunniteltiin ja toteutettiin alusta loppuun automaattinen laiteohjelmiston testausjärjestelmä Epec Oy:n valmistamia koneenohjausyksiköitä varten. Työ koostui testausjärjestelmän määrittelystä, suunnittelusta, rakentamisesta, ohjelmoinnista ja järjestelmän testauksesta. Nämä kaikki vaiheet on toteutettu tämän työn aikana ja lopputuloksena on syntynyt tarkoituksen mukainen koneenohjausyksikön laiteohjelmiston automaattinen testausjärjestelmä.

6.2. Alkuperäiset tavoitteet

Tämän työn alkuperäinen tavoite oli tuottaa Epecin ohjausyksiköiden laiteohjelmiston testausongelmaan ratkaisu, jolla saavutetaan riittävän hyvin testaukselle asetetut kriteerit. Testaukselle asetettujen kriteerien saavuttamiseksi oli laiteohjelmiston toiminta voitava varmentaa oikeelliseksi kaikkien sen sovellusohjelmoijalle tarjoamien prosessoriin, muistin käsittelyyn ja kommunikaatioon liittyvien ominaisuuksien osalta. Testauksen riittävän hyvän laadun ja toistettavuuden saavuttamiseksi tarkoituksena oli rakentaa testausjärjestelmä, jolla voidaan suorittaa kattavia testejä ohjausyksiköiden laiteohjelmiston testaamiseksi niin, että laiteohjelmiston laatu vastaa luotettavasti sille asetettuja vaatimuksia.

Tämän diplomityön tuloksena suunniteltiin ja rakennettiin automaattinen koneenohjausyksiköiden laiteohjelmiston testausjärjestelmä, jolla voidaan suorittaa automaattisia testejä ohjausyksiköiden laiteohjelmistolle kattaen kaikki aiemmin manuaalisesti suoritetut testit. Tämänhetkinen testausjärjestelmä kattaa kaikki laiteohjelmiston toiminnot perustason testeillä. Testien monipuolisempi laadunvarmennus ja yksityiskohtaisempi toimintojen testaus tullaan toteuttamaan jatkokehitysprojekteissa tämän diplomityöprojektin jälkeen. Kuitenkin jo perustason testeillä ylletään parempaan laadunvarmennukseen ja testien toistettavuuteen kuin manuaalisilla testeillä ennen automaattista testausjärjestelmää. Parempi laadunvarmennus saavutetaan jo perustason testeillä, koska esimerkiksi testausraporttiin voidaan helposti kerätä paljon enemmän informaatiota kuin manuaalisen testauksen aikana ja jo perustasolla automaattiset testit ovat paljon kattavampia kuin aiemmin manuaalisesti suoritetut testit.

6.3. Alkuperäiset vaatimukset

Tärkeimpänä alkuperäisenä vaatimuksena testausjärjestelmälle oli sen modulaarisuus. Testausjärjestelmän tuli olla helposti laajennettavissa niin laitteiston osalta kuin myös ohjelmistollisesti. Testausjärjestelmällä tuli voida suorittaa erilaisten ohjausyksiköiden laiteohjelmiston testaus automaattisesti ilman, että järjestelmää itsessään olisi täytynyt muuttaa.

Alkuperäisten vaatimusten mukainen testausjärjestelmän laitteiston modulaarisuus toteutettiin PXI-arkkitehtuurin avulla. PXI-arkkitehtuuri varmistaa laitteiston helpon ja nopean laajennuksen tulevaisuuden tarpeiden mukaan. PXI-kehikoita on mahdollisuus ketjuttaa yhteen, joten tila uusilta I/O-korteilta ei tule loppumaan kesken. Testausjärjestelmän ohjelmiston modulaarisuus on myös korkeaa luokkaa. Ohjelmiston yksittäiset ohjelmamoduulit toteutettiin LabVIEW:llä siten, että niitä voidaan uudelleen käyttää erilaisten ohjausyksiköiden testauksessa muuttamalla ohjelmamoduulien parametreja. Myös TestStand:llä toteutettu testaussekvenssi on modulaarinen, jolloin testaussekvenssiin voidaan tulevaisuudessa lisätä testausohjelmia mihin tahansa kohtaan. Sekvenssin alatason osat eivät ole riippuvaisia toistensa toiminnoista tai datasta, vaan ne toimivat yksittäisinä testausyksiköinä riippumatta toisistaan, mikä varmistaa sekvenssin modulaarisuuden.

Toinen keskeinen vaatimus testausjärjestelmälle oli sen automaattisuus. Testien automatisoinnilla oli tarkoitus saada aikaan testitapauksin helppo suoritus sekä toistettavuus. Samankaltaisia testejä tuli voida toistaa useita kertoja useille erilaisille ohjausyksiköille. Automatisoinnin myötä oli tarkoitus lyhentää kokonaistestausaikaa huomattavasti. Tarkoitus oli myös kyetä suorittamaan automaattisesti testejä, jotka manuaalisella testauksella eivät ole mahdollisia.

Testausjärjestelmän automaattisuuden taso on jo varsin korkea ja sitä tullaan vielä lisäämään jatkokehitysprojekteissa. Testitapaukset toimivat automaattisesti testaajasta riippumatta siten, että kaikki testisekvenssin testausosat suoriutuvat automaattisesti. Sekvenssin toiminta vaatii, että ohjausyksikköön on asennettu laiteohjelmisto ja testaussovellus valmiiksi manuaalisesti. Lisäksi ohjausyksikön asetukset täytyy asettaa manuaalisesti oikeiksi ennen testauksen aloittamista. Nämä manuaaliset toiminnot on mahdollista automatisoida ja ne myös tullaan automatisoimaan myöhemmin. Tiettyjen toimintojen manuaalinen suoritus on kuitenkin vielä tässä vaiheessa järkevää, jotta niiden automaattinen toteutus ei olisi vienyt aikaa tärkeimmältä eli testaamisen automatisoinnilta. Testisekvenssiä voidaan ajaa suoraan haluttu toistomäärä silmukassa tai vaihtoehtoisesti käynnistämällä se aina manuaalisesti uudestaan, jolloin samat testit toistuvat samaan tapaan kuin silmukassa ajettaessa. Testisekvenssi on aina ajettaessa sama, joten testien toistettavuus, verrattaessa manuaaliseen testaukseen, on paljon parempi. Automaattisen testausjärjestelmän avulla ohjausyksiköiden laiteohjelmiston testausaika on nykytasolla arviolta joitakin prosentteja (5–10 %) verrattaessa manuaalisen testauksen kuluttamaan aikaan.

6.4. Ratkaisuvaihtoehdot

Ratkaisuvaihtoehtoina tässä työssä oli suunnitella ja toteuttaa testausjärjestelmä itse kokonaan tai vaihtoehtoisesti hankkia jokin markkinoilla tarjolla oleva valmis järjestelmäkokonaisuus ja tehdä ainoastaan ulkoiset liitynnät ja ohjelmisto itse. Projektissa päädyttiin jo melko aikaisessa vaiheessa suunnittelemaan ja rakentamaan koko järjestelmä itse. Jo pelkästään se, että järjestelmän kokonaiskustannukset haluttiin pitää mahdollisimman pieninä, johti järjestelmän omaan kokonaistoteutukseen. Jos testausjärjestelmällä olisi ollut hirveä kiire, olisi laitteisto varmasti jouduttu hankkimaan ulkopuolisilta toimijoilta, jolloin koko projektin kesto olisi ollut huomattavasti lyhyempi. Koska järjestelmän toteutukseen ja suunnitteluun oli kuitenkin riittävästi aikaa, oli itse kokonaan toteutettu järjestelmä hyvä ja toimiva ratkaisu.

6.5. Tulevaisuus

Tässä työssä toteutettu testausjärjestelmä on diplomityöprojektin päätyttyä valmis automaattinen testauskokonaisuus ohjausyksiköiden testaukseen. Koneenohjausyksiköt ovat kuitenkin toimintaympäristöltään ja tyypiltään sellaisia laitteita, että uusia testattavia erityistoimintoja löytyy niistä väkisinkin lisää. Tulevaisuudessa järjestelmää joudutaankin laajentamaan uusien tulevien tuotteiden mukaiseksi. Jatkossa tärkeää roolia testauksessa esittää testattavien erikoistoimintojen arviointi. Automaattiseen testausjärjestelmään ei lopulta kuitenkaan ole mielekästä toteuttaa liian yksityiskohtaisia testejä, joita ei toisteta useita kertoja erilaisille testiobjekteille. Tällaiset yksityiskohtaiset, jopa vain kerran toistettavat testit, on jatkossakin mahdollisuuksien mukaan järkevää toteuttaa manuaalisesti. Suurin muutos tulevaisuudessa on varmasti automaattisen testausjärjestelmän mahdollistama pitkäaikaisten testien suoritusmahdollisuus sekä taustatoimintojen lisäyksen mahdollisuus testeihin, jolloin testien suoritus vastaa lähemmin ohjausyksiköiden todellista käyttötapaa.

Testausjärjestelmän automaattisuuden lisäys on myös yksi tulevaisuudessa toteutettava kehityskohde. Järjestelmästä on mahdollista tehdä lähes täysin automaattinen, poislukien tietenkin testiobjektien kytkentä järjestelmään. Kokonaistestausajan pituuden vähentäminen onnistuukin huomattavasti automaattisuuden tason nostolla. Kun laiteohjelmiston kokonaistestausaikaa saadaan lyhennettyä, lyhenee myös laiteohjelmiston kokonaiskehitysaika, jota kautta uuden laiteohjelmistoversion julkaisemisaika lyhenee. Yleensä uuden laiteohjelmistoversion kokonaiskehitysaika onkin varsin kriittinen ja se halutaan minimoida mahdollisimman lyhyeksi huonontamatta kuitenkaan laiteohjelmiston laatua.

Järjestelmää laajennettaessa, lisättäessä sen toimintoja ja uusia testitapauksia toteutettaessa on muistettava laitteiston ja ohjelmiston modulaarisuuden ylläpito. Järjestelmän modulaarisuus mahdollistaa nopean ja joustavan jatkokehityksen pitkälle tulevaisuuteen, kunhan modulaarisuus vaatimuksista ei tingitä missään vaiheessa järjestelmän jatkokehitysprojekteja.

7. Yhteenveto

Tässä työssä suunniteltiin ja toteutettiin automaattinen koneenohjausyksiköiden laiteohjelmiston testausjärjestelmä. Työ aloitettiin määrittelemällä järjestelmältä vaaditut toiminnot ja tätä kautta laitteistovaatimukset. Laitteistovaatimuksien perusteella hankittiin kaikki tarvittavat laitteet ja ohjelmistot kokonaisvaltaisen automaattisen testausjärjestelmän toteuttamiseksi. Laitteiston hankinnan jälkeen järjestelmälle määriteltiin testaussekvenssi, jonka avulla toteutettuja yksittäisiä testejä voidaan ajaa järjestelmällisesti niin, että kaikki halutut toiminnot tulee testatuksi. Testaussekvenssin määrittelyn yhteydessä määriteltiin myös yksittäiset toteutettavat ohjelmamoduulit, jotka vastaavat laiteohjelmiston toimintojen yksittäisistä testeistä. Työn yhteydessä suoritettiin myös laitteiston kokonaisvaltainen testaus, ohjelmamoduulien yksikkötestit sekä koko ohjelmiston integraatiotestaus.

Työn kirjallisessa osuudessa on esitelty aluksi testausjärjestelmän tarpeeseen vaikuttaneet tekijät ja järjestelmän toteutuksen lähtökohdat. Seuraavaksi on käyty läpi kirjallisuuskatsauksessa testausjärjestelmällä testattavien koneenohjausyksiköiden ominaisuuksia laiteohjelmiston toiminnasta sen ohjaamiin yksittäisiin I/O-toimintoihin ja kommunikaatioon. Testiobjektiin perehtymisen jälkeen on tarkasteltu tämän työn kannalta tärkeää testauksen teoriaa ja testauksen automatisoinnin erityispiirteitä. Tämän jälkeen on perehdytty automaattiseen testauslaitteistoon ja sen ohjelmointiin. Kirjallisuuskatsauksen jälkeen on esitelty testausjärjestelmän toteutus. Lopuksi on vielä tarkasteltu järjestelmän käyttöä, ylläpitoa ja elinkaarta sekä pohdittu työn onnistumista ja järjestelmän tulevaisuutta.

Lähtökohtana oli toteuttaa ratkaisu koneenohjausyksiköiden testausongelmaan. Ongelmana oli, että ohjausyksiköiden laiteohjelmiston testaukselle asetettuja laatu- ja toistettavuusvaatimuksia ei enää saavutettu manuaalisen laiteohjelmiston testauksen keinoin. Tämän ongelman pohjalta työlle luotiin alussa vaatimukset ja tavoitteet. Ongelmaan toteutettiin tämän työn aikana vaatimusten ja tavoitteiden mukainen ratkaisu, joka oli automaattinen laiteohjelmiston testausjärjestelmä koneenohjausyksiköille.

Alkuperäisten tavoitteiden ja vaatimusten perusteella työtä voidaan pitää mallikkaasti onnistuneena. Testausjärjestelmän pidempiaikainen käyttö tulee kuitenkin vasta näyttämään järjestelmän todelliset edut ja myös mahdolliset puutteet. Oikein käytettynä järjestelmä ja sen tuottamat testausraportit tarjoavat todella paljon uutta lisäarvoa ohjausyksiköiden suunnitteluprosessiin, muun muassa nopeutuneena ja parantuneena kokonaistestausprosessina. Järjestelmää käytettäessä täytyy kuitenkin aina muistaa, että se on vain kone, joka tekee mitä se on ohjelmoitu tekemään. Tässä työssä on valettu pohja testausjärjestelmälle, jota kehittämällä voidaan testauksen kattavuutta, nopeutta ja luotettavuutta parantaa koko ajan paremmalle tasolle. Testausprosessin jatkuva kehittäminen tuleekin olemaan suuri osa jatkossa tapahtuvasta testausjärjestelmän ylläpitoprosessista.

Hyvin ylläpidettynä ja kehitettynä testausjärjestelmän tuottama lisäarvo saadaan maksimoitua pitkän elinkaaren muodossa. Olennaisia elinkaareen vaikuttavia seikkoja ovat laitteiston ja ohjelmiston modulaarisuuden säilyttäminen sekä pitkällä tähtäimellä tehty kehitystyö, joka ottaa huomioon niin jo olemassa olevat testiobjektit kuin myös tulevaisuudessa testattavat uudet kohteet. Testausjärjestelmää tullaan tulevaisuudessa laajentamaan uusien tuotteiden testaukseen. Siitä johtuen jo tätä työtä tehtäessä on pyritty toimimaan mahdollisimman paljon tulevaisuuden laajennuksia silmälläpitäen. Tämä näkyy muun muassa laitteiston ja testaussekvenssin modulaarisuutena sekä yksittäisten ohjelmamoduulien uudelleenkäyttömahdollisuuksissa.

Viitteet

[1] Leea Hiltunen; Petri Mäkikyrö; Antti Rantamäki, Ohjelmoitavat logiikat, Seminaariesitelmä 2004, Teknillinen korkeakoulu, Konetekniikan osasto, Tuotantoautomaatio, Espoo.

[2] Kari Koskinen, Hajautetun koneenohjausjärjestelmän sisäinen ja ulkoinen tietoliikenne, Luentomateriaali 2008, Teknillinen korkeakoulu, Konetekniikan osasto, Mekatroniikka ja hydrauliikka, Espoo.

[3] Anon., Epec Oy, Päivitetty: 09.11.2008, Viitattu: 09.11.2008, Saatavissa: http://www.epec.fi/index.html.

[4] Jarmo Alanen, CAN – ajoneuvojen ja koneiden sisäinen paikallisväylä, Opetusmateriaali 2003, VTT Tuotteet ja tuotanto, Älykkäät koneet ja palvelut, Tampere.

[5] Wikipedia, Ohjelmoitava logiikka, Päivitetty: 09.09.2008, Viitattu: 09.11.2008, Saatavissa: http://fi.wikipedia.org/wiki/Ohjelmoitava_logiikka.

[6] Wikipedia, Firmware, Päivitetty: 31.10.2008, Viitattu: 09.11.2008, Saatavissa: http://fi.wikipedia.org/wiki/Firmware.

[7] Anon., Epec 2024 Universaali I/O-moduuli, Datalehti 2008, Ei julkisesti saatavilla.

[8] Hugh Jack, Automating Manufacturing Systems with PLCs, Versio: 5.0, GNU Free Documentation Licence, 2007.

[9] Robert Lewis, Programming Industrial Control Systems Using IEC 1131-3, Englanti, Institution of Electrical Engineers, 1998, ISBN: 9780852969502.

[10] Anon., PLCopen for efficiency in automation, Päivitetty: 21.10.2008, Viitattu: 09.11.2008, Saatavissa: http://plcopen.org/.

[11] Anon., CoDeSys, Päivitetty: 09.11.2008, Viitattu: 09.11.2008, Saatavissa: http://www.3s-software.com/.

[12] Nebojsa Matic, Introduction to PLC controllers, Päivitetty: 09.11.2008, Viitattu: 09.11.2008, Saatavissa: http://www.mikroe.com/en/books/plcbook/plcbook.htm.

[13] Anon., I/O Devices, Päivitetty: 09.11.2008, Viitattu: 09.11.2008, Saatavissa: http://www.plctutor.com/plc-io-devices.html.

[14] Anon., National Instruments 651x User Manual, Käyttäjämanuaali 2007, Viitattu: 09.11.2008, Saatavilla: http://www.ni.com/pdf/manuals/372172a.pdf.

[15] Wikipedia, Pulssinleveysmodulaatio, Päivitetty: 30.10.2008, Viitattu: 09.11.2008, Saatavissa: http://fi.wikipedia.org/wiki/Pulssinleveysmodulaatio.

[16] Veijo Hänninen, Elektroniikkaa koneenrakentajalle, Prosessori-lehti, Helmikuu 2/2008, Sivut 34-37.

[17] Konrad Etschberger, Controller Area Network – Basics, Protocols, Chips and Applications, Englanti, IXXAT Automation GmbH, 2001, ISBN: 3000073760.

[18] CiA Draft Standard 301, CANopen Application Layer and Communication Profile, Versio: 4.02, CAN in Automation, 2002, 135 s.

[19] Wikipedia, CANopen, Päivitetty: 21.10.2008, Viitattu: 09.11.2008, Saatavissa: http://en.wikipedia.org/wiki/CANopen.

[20] ANSI/TIA-232-F-1997 (R2002), Standardi TIA-232, Versio: F, Telecommunications Industry Association, 2002, 51 s.

[21] ANSI/TIA-485-A-98 (R2003), Standardi TIA-485, Versio: A, Telecommunications Industry Association, 2003, 26 s.

[22] Bart Broekman; Edwin Notenboom, Testing Embedded Software, Englanti, Addison-Wesley Professional, 2002, ISBN: 9780321159861.

[23] Anon., LabVIEW Basics I: Introduction Course Manual, Kurssimateriaali 2007, Ei julkisesti saatavilla.

[24] LabVIEW Wiki, LabVIEW, Päivitetty: 26.05.2008, Viitattu: 09.11.2008, Saatavissa: http://wiki.lavag.org/LabVIEW#Learning_LabVIEW.

[25] Wikipedia, LabVIEW, Päivitetty: 09.11.2008, Viitattu: 09.11.2008, Saatavissa: http://en.wikipedia.org/wiki/LabVIEW.

[26] Anon., LabVIEW, Päivitetty: 09.11.2008, Viitattu: 09.11.2008, Saatavissa: http://www.ni.com/labview/.

[27] Anon., TestStand, Päivitetty: 09.11.2008, Viitattu: 09.11.2008, Saatavissa: http://www.ni.com/teststand/.

[28] Anon., Pickering 40-615 Very High Density Multiplexer Module, Datalehti 2008, Viitattu: 09.11.2008, Saatavilla: http://www.pickeringtest.com/pdf/40-615D.pdf.

[29] Veijo Hänninen, Suomalaisia PXI-sovelluksia, Prosessori-lehti, Huhtikuu 4/2008, Sivut 46-47.

[30] Anon., PXI, Päivitetty: 09.11.2008, Viitattu: 09.11.2008, Saatavissa: http://zone.ni.com/devzone/cda/tut/p/id/4811.

[31] Anon., PXI Systems Alliance, Päivitetty: 09.07.2008, Viitattu: 09.11.2008, Saatavissa: http://www.pxisa.org/Overview.html.

[32] Phillip A. Laplante, Real-Time Systems Design and Analysis, 3. Painos, USA, IEEE COMPUTER SOCIETY PRESS, 2004, ISBN: 9780471228554.